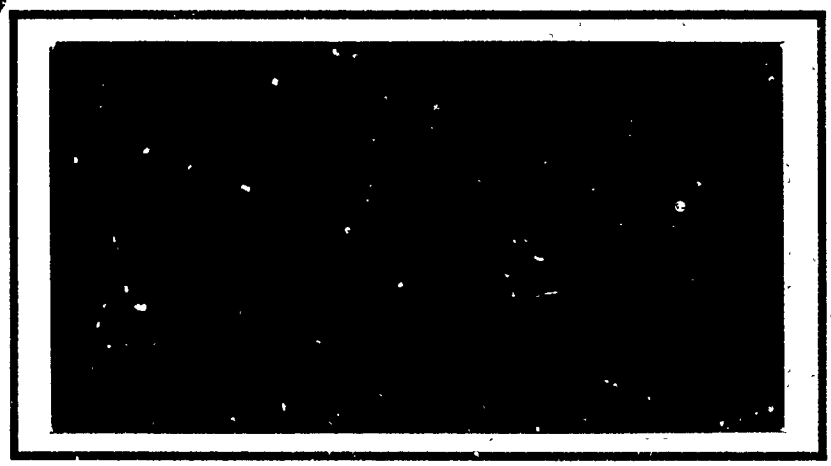




DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

# AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/90D-47



# ANALYSIS OF VISUAL ILLUSIONS USING GABOR FILTERS

THESIS

Richard A. Oberndorf
Captain, USAF

AFIT/GE/ENG/90D-47

AFIT/GE/ENG/90D-47

# ANALYSIS OF VISUAL ILLUSIONS USING GABOR FILTERS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Richard A. Oberndorf, B.S.
Captain, USAF

December, 1990





## *Acknowledgments*

My gratitude and appreciation go out to all those whom I have been able to learn from regarding the compilation of this document. To the Graphics Weenies (Dave Dahn, Phil Platt, Ed Williams, Buck Stuart, Bill DeRouchey), thanks for letting me tie up Dali as much as I did. Dave, thanks for getting me from RAS to RLE to 2DFFT to IFF to RLE to EPS and on to paper. Ed, thanks for the brick wall. Rick Ricart deserves honorable mention for pointing me from Mathematica towards psfix – a very Gaboristic accomplishment.

Much thanks also goes to Maj. Phil Amburn for putting up with abused SUN manual pages, tossed aside video screens, unattended terminals, and for always solving my computer difficulties with that infamous phrase, "Well, let's look at the man page for that." I'm now firmly convinced that the difference between being a Master's Student and a Master's Graduate is not how *much* you know but that you know where to find it.

Dr. Matt Kabrisky also played a important part in this effort coming to fruition. From his intuitive analysis of a portion of the background research ("Why go to the books, when you have a dinosaur in your midst?"), to his in-depth and totally objective viewpoints on fellow researchers ("...that no good SOB."); from his occasional wettings, to his pats-on-the-back when things looked bleak ("You're o.k. Rick ...I don't care what Steve says.") — anyone who fields such diverse questions in class (such as, "Do we know anything about cell organization in the higher areas of the brain?" and "I recently read a report about people with three eyes ...") and responds with equal enthusiasm, is alright in my book.

Dr. Steve Rogers was the thesis advisor for this project; as such, he kept me focused on the tasks at hand when I might have strayed too far off the path in searching for 'Kanizsa Karma'; he challenged me by not only answering the questions I posed, but by taking it a step further by giving me two more questions to think about. The only way to describe his method of teaching is that I have never met anyone who enjoys *thinking* as much as Steve; whether it was in front of a class, or in our weekly meetings there seemed to be a sincere

joy in contemplating whatever problem/question came up. Most importantly, he taught me that you're never too old to dive for a softball ... it's just that not as many wind up in your glove.

When you throw two strong willed individuals into a tiny apartment, tell them that you'll see them in eighteen months, and that they should try to have fun, you're asking for trouble. Especially when one of them is a ten year old who *thinks* she's going on thirty-three, and the other, by all rights, should be a Governor at this point in her career. Not too remarkably though, this didn't happen with my best friend Gina and our daughter Courtney. I say 'not too remarkably' because they have consistently supported each other and myself throughout our Air Force lives. It hasn't always been easy, but it has always been fun. I love you guys.

Finally, as with any good Air Force operation, there are support troops which need to be thanked. Ray, for helping me maintain that sophisticated sense-of-humor; the AFIT-EN softball team for teaching me to *never, ever* again say that I had played infield in the past; Debbie, for helping Gina maintain; the Detroit Connection, for providing a flop house and M&Ms; and, lastly, to the MILSTAR program ... ready or not folks, here I come.

Richard A. Oberndorf

# *Table of Contents*

Page

## *List of Figures*

*Abstract*

This thesis has demonstrated the correctness of using spatial filters in the analysis of various visual illusions; the specific form of filter has been derived from a Gabor equation model of simple cell response on the visual cortex. The Gabor Low Pass Filter (GLPF) applied to these anomalies was derived from proposals made that simple cell response on the visual cortex may be modeled by a set of equations originally proposed by Gabor in the 1940's. Based upon the extension of these equations into two dimensions, a GLPF process was applied to computer-generated black and white illusions (the Kanizsa Triangle, the Spoked Circle and the Ehrenstein Illusion). The results demonstrate that the anomalous contour present in these illusions are explained by an energy boundary surrounding the anomalous area. These differing energies are a direct result of the GLPF process.

# ANALYSIS OF VISUAL ILLUSIONS USING GABOR FILTERS

## *I. Introduction*

Much work has been accomplished attempting to understand how and why the human brain perceives visual illusions. From a physiological viewpoint, the process of vision may be divided into two main operations. The first of these deals with the processing which takes place when light first enters the eye up to the point when this information reaches the visual cortex of the brain. The second area regarding the concept of vision is concerned with how the brain uses this data—the cognition and conceptual formulation which allows us to recognize objects. Before claiming to understand fully the inner workings of this second region, one should first know what form the data is in after being processed by the first region. This report looks at a viable way to model this processed data, via Gabor equations, and then turns toward possible explanations of certain categories of visual illusions.

Why concern ourselves with visual illusions? In the field of pattern recognition, if one truly desires to create a machine which can emulate this first area of human vision, understanding these visual anomalies becomes a necessity. Note the use of the word 'anomaly' ("deviation from the common rule: IRREGULARITY" (33:46)) vice 'illusion' (" a misleading image presented to the vision: something that deceives or misleads intellectually" (33:566)). This will be prevalent throughout the remainder of this document; the contention here is that there is no deception or misleading involved with these figures – the explanations rely on the processing previously mentioned taking place prior to the higher, or conceptual, regions of the brain. The root of this paper, its underlying theme regarding the idea of visual anomalies, can be taken from the science of philosophy:

> [Man's] senses do not provide him with automatic knowledge in separate snatches independent of context, but only with the material of knowledge, which his mind must learn to integrate. ... His senses cannot deceive him, ... physical objects cannot act without causes, ... his organs of perception are physical and have no volition, no power to invent or to distort ... the evidence they give him is an absolute, but his mind must learn to understand it, his mind must discover the nature, the causes, the full context of his sensory material, his mind must identify the things that he perceives.(39:156)

The first chapters of this thesis deal with the background work which has led to Gabor filters being used as models of simple cell response in the visual cortex, and previous work done by Ginsburg in the evaluation of visual anomalies. A description of the equipment and methodology to be used will be provided in Chapter 4, followed by the purpose of this research effort — the use of Gabor filters to support, and perhaps to expand upon, previous Fourier based models. All applicable software is provided in the appendices of this document.

# *II. Work Leading to Gabor Filter Models*

## *2.1 Initial Experiments*

In 1962, Hubel and Wiesel, two physiologists working at Harvard Medical School, conducted experiments designed to measure cell response in the cat's visual cortex (18). The visual cortex of a mammal is located at the lower, rear portion of the brain and makes up a large portion of the surface area there. This area was chosen for study due to its diversity of cell types and interconnections (18:106). Figure 1 shows the top view of a cat's visual cortex (shaded), while Figure 2 provides a look at the visual system (shaded) from eye to cortex with a bottom-up perspective (17).

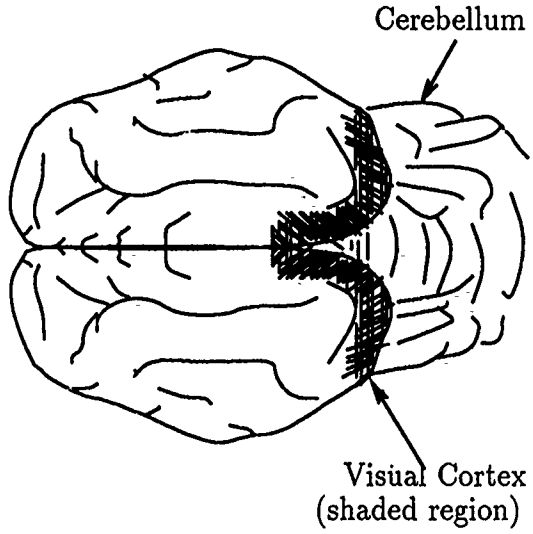



Figure 1. Top View of Cat's Visual Cortex

*2.1.1 Methodology.* Forty anesthetized (injections of thiopental sodium) cats were used, their eyes stabilized with succinylcholine, and pupils dilated with atropine. While stimulating and recording methods will not be detailed here (may be found in (16)), the

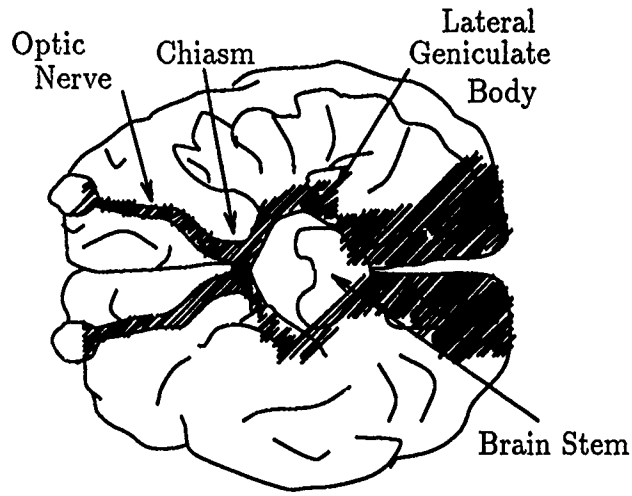



Figure 2. Bottom View of Cat's Visual System

preparation method used on the cats has been provided to point out that some alterations had to be made in recording data from the animals. For example, the eyes generally diverged slightly due to the muscle relaxant used which affected the centers of gaze. Other factors influenced by the preparations used were inward eye rotation and angular displacements of the receptive fields. The animals were placed 1.5 m from a screen where various patterns of white light were shone. The receptive fields of each cell were mapped out separately for the two eyes on sheets of paper (18:107).

A micro-electrode was introduced into the cortex, with penetrations of no deeper than 3 or 4 mm. Figure 3 shows the approximate location of the 45 points of entry into the cortex. Lesions were made for each penetration, small enough to indicate the position of the electrode tip to the nearest cortical layer (18:108).

*2.1.2 Results of Cell Monitoring.* Hubel and Wiesel defined the receptive field of a cell in the visual system as the region of the visual field over which one can influence the firing of that cell. They noted that cortical cells differed in the complexity of their receptive

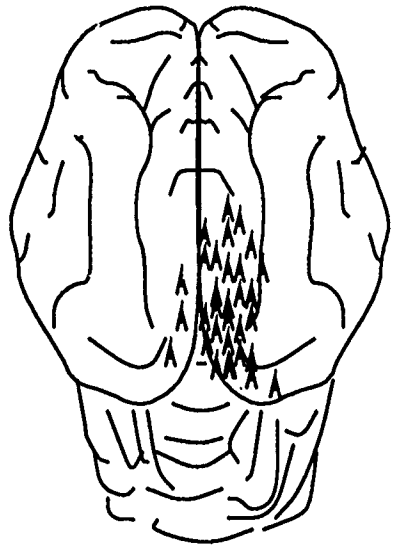


Figure 3. Entry Points of Micro-Electrode Penetrations(18:108)

fields; two main groupings were labeled as 'simple' and 'complex'; however, they were quick to note that new classifications of receptive fields should soon be revealed – a point borne out by a later discovery known as 'hypercomplex' cells (19). Their discussion was limited to the simple and complex cell classifications (18:109).

*2.1.2.1 Simple Receptive Fields.* The 'simple' classification was assigned to 233 of the 303 cortical cells observed; these all possessed distinct excitatory and inhibitory sub-regions, i.e., illuminating part or all of an excitatory region maintained cell firing, while a light shone in an inhibitory region suppressed firing. While the cortical receptive fields were similar to retinal ganglion and geniculate cells in possessing excitatory and inhibitory sub-regions, the spatial arrangements of these areas differed. The geniculate cells demonstrated a concentric organization (Figure 4-A), while the cortical cells had straight-line boundaries (Figure 4-B). It was noted that the most effective stationary stimulus for these type of cells was a long narrow slit of light just large enough to cover the central excitation region.

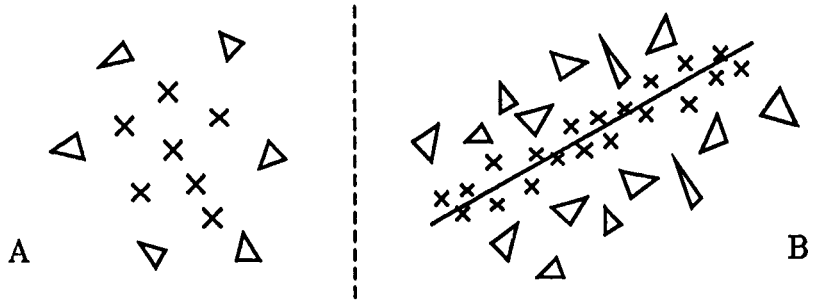



Figure 4. Arrangement of Geniculate (A) and Cortical Receptive Fields (B); 'x' indicates excitatory response; △ indicates inhibitory response (18:111)

As a side note for now, but one that will become more important once modeling of cell response is discussed (spatial vs. spatial frequency dependency), the researchers note that "For maximum centre response the *orientation* of the slit was critical; changing the orientation by more that $5° - 10°$ was usually enough to reduce a response greatly or even abolish it " (18:111) . Also, moving stimuli were very effective in generating cell response; "With moving stimuli, just as with stationary, the *orientation* was critical" (18:112).

*2.1.2.2 Complex Receptive Fields.* Those cells labeled 'complex' were just that; there was no straight-forward method to categorize the response. Four activators were used for these cells: a slit of light with a non-uniform field, a slit with a uniform field, an edge, and a dark bar. The cells responded to each activator with some semblance to a simple cell, yet generally responded to all four activators where a simple cell would respond to only one type. They did note that features typical of complex cells were a lack of summation among these responses and a wide area within the receptive field in which the activator was effective (18:120). Perhaps the major difference between the two classifications was that a stimulus presented to a complex cell was effective wherever it was placed in the receptive field, provided the orientation was appropriate, whereas the simple cell required stimulus placement either within a specific excitatory or inhibitory region (18:151).

*2.1.3 Functional Architecture.* By comparing responses of cells recorded in sequence, observing the unresolved background activity, and comparing cells recorded simultaneously, the functional architecture of the visual cortex was evaluated. The cortex had a columnar organization, where each cell within the column had the same receptive field and orientation specificity. The columns appeared to have an approximate diameter of 0.5 mm and extended from surface to white matter (18:152).

The concepts of receptive fields and the columnar organization of the cortex became extremely important to further research in modeling the visual system and will be discussed in detail in later sections.

## *2.2 Differing Opinions on Experimental Results*

Toward the later part of the 1960s, many researchers began voicing their concerns that simple cells were not *spatially* dependent, as claimed, but spatial-*frequency* dependent. That is, Hubel and Wiesel stressed the dependency of the cells on the orientation of the stimulus and claimed that to be proof that the simple cells were dependent only on where the stimuli were placed in the receptive field, while the width and orientation of the stimuli were leading engineers toward evaluating the frequency characteristics of the input when searching for an ideal model for cortical cell response. A more precise way of stating this conflict comes directly from an article by Daugman:

> Both perceptual and neurophysiological vision research in the past two decades have been enlivened by debate over whether the fundamental character of early visual representation involves space-domain local feature detection or more closely resembles a Fourier-like decomposition into spatial-frequency components. (8:1160)

Both sides of the debate conducted a number of experiments, all of which seemed to have substantive arguments for that particular point of view (8:1160). For every Hubel and Wiesel experiment backing their original results (20), there seemed to be a Maffei and Fiorentini research effort which staked a strong claim for the spatial-frequency point of view (31). Figure

5 (31:1257-1259) provides perhaps the most noteworthy outcome from Maffei and Fiorentini's work, response to variations of spatial frequency in retinal (A), lateral geniculate nucleus (LGN, B), and simple (C) cells. While Figure 5 shows only a few examples of each cell response, it is apparent that there is a "progressive narrowing of the response curves from the retina to the LGN and to the simple cortical cells [which] suggests that these are three successive stations in the process of elaboration of spatial information" (31:1260). A point to note regarding Figure 5-C: the shape of the response curves is very similar to a Gaussian envelope which, as will be described in later sections detailing Gabor filters, is the shape of a Gabor filter in the spatial frequency domain.

However, many researchers were not "debaters" of this issue—they simply sought to clarify what was occurring within the visual cortex (1) (9) (30). Many of these experimenters came to the conclusion that it was not a space *vs.* frequency issue but rather a space *and* frequency problem. For example, Albrecht, De Valois, and Thorell evaluated 96 simple cells in both monkey and cat visual cortices and found greater sensitivity to the adjustments of frequency characteristics of the light than spatial variations in all 96 cases... but they did point out that there was sensitivity to both (1:89). These types of studies led to a wave of research (28) (29) (32) that proved both space and frequency representations of simple cell responses were correct. At the root of the arguments presented in these papers was the claim that simple cells behave, for the most part, linearly.

### *2.3 The Linear Cell*

The total contribution of activated cells within the receptive fields of the visual cortex is a linear one (35). A linear mechanism is characterized by the fact that selectivity in either the space or frequency domains implies a complementary selectivity in the other domain. Initially, Hubel and Wiesel (18) stated that simple cells *may* behave linearly; in later years, one effort claimed that the modulation depth of gratings is proportional to the logarithm of the grating's constant (31). However, it was shown (36) that the amplitude of the modulated response of both simple and complex cells appear to be linearly related to contrast. Andrews

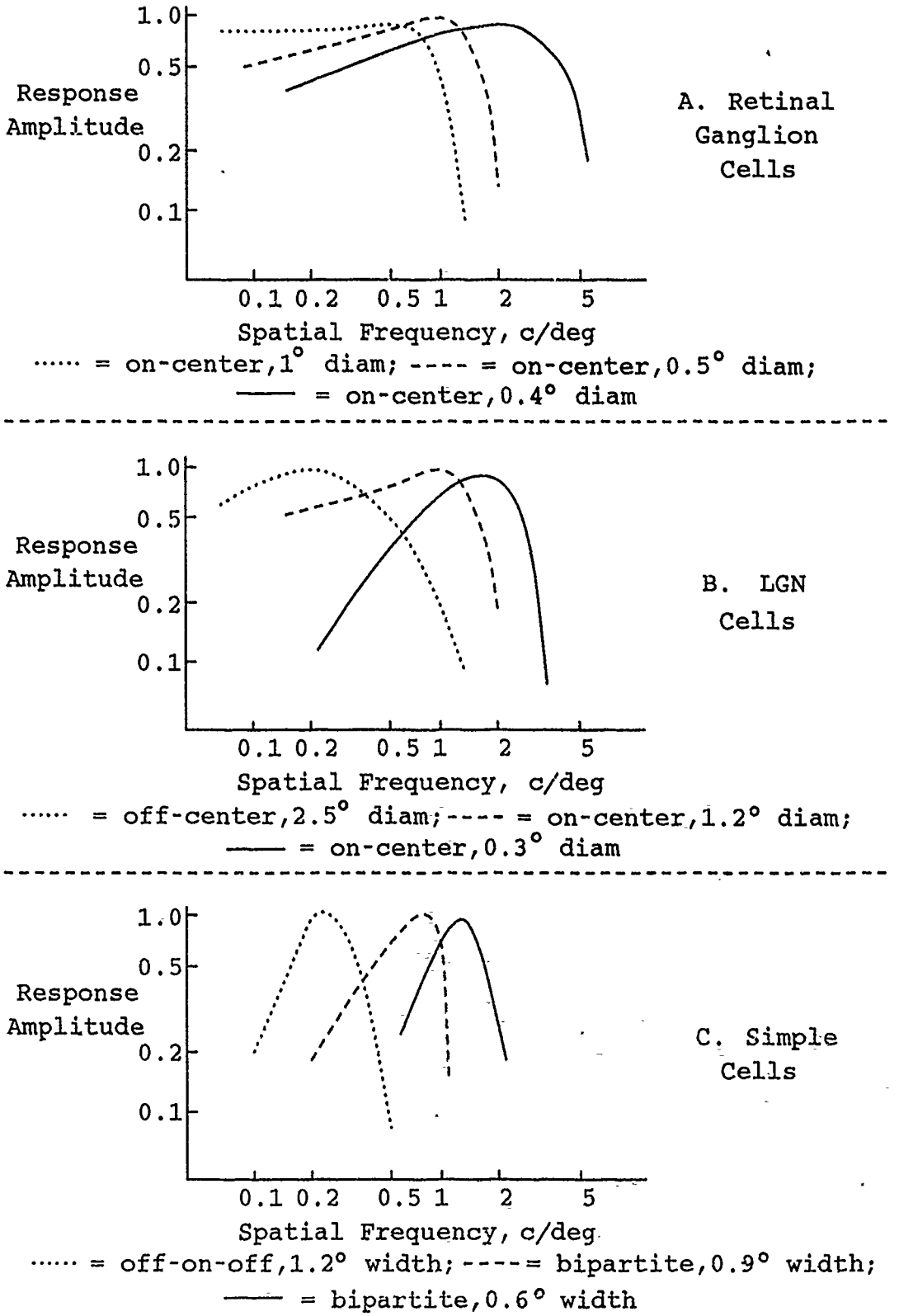



Figure 5. Cell Response to a Sinusoidal Grating of Varying Spatial Frequency. Width/Diameter Refer to Cell Receptive Field (31:1257-1259)

and Pollen (2) substantiated this claim by demonstrating how a simple cell's response to a sinusoidal stimulus reflects how the cell performs linear spatial summation. This linearity was the key point in finally reaching conciliation between the two sides. The acceptance of the complimentarity of both descriptions was described as being reminiscent of the eventual dissolving of the historic wave vs. particle debate in quantum physics (8:1160).

Research during this time (1980–1981) began stressing the appropriateness of working in both domains (28) (29) (32). One of these efforts, Marčelja's work in 1980, was particularly important in that it was one of the first to mention Gabor's work in the field of communications signal theory as a possible model for the biological processes taking place in the visual cortex (32). This communication theory will be reviewed in the following section.

### *2.4 Gabor's Theory of Communication*

Gabor referenced what was at that time (1946) the practice of describing a signal with either samples from sharply defined instants of time *or* rigorously defined frequencies. But many of our sensations require a description in both time *and* frequency—especially the auditory sensations (11:429). Although written to assist in the analysis of hearing sensations, Gabor's paper applies to the transmission of data in general.

Emphasizing the work which "dawned gradually upon communication engineers" (11:429) during the 1920s, Gabor based his theory on the fact that transmitting a specific amount of information per unit time requires a minimum spectral bandwidth. Using a form of the Schwarz Inequality credited to Weyl and Pauli, Gabor stated the uncertainty relation (sometimes referred to as the space-bandwidth product) as(11:434):

$$\Delta t \cdot \Delta f \geq \frac{1}{2} \tag{1}$$

More importantly, he determined that a harmonic oscillation enveloped by a gaussian function met the lower bound of $\Delta t \cdot \Delta f = \frac{1}{2}$. This Gabor function, $\psi(t)$, can be written as

(11:435):

$$\psi(t) = e^{-\alpha^2(t-t_o)^2} \cdot e^{j2\pi(f_o t+\phi)} \tag{2}$$

where the first exponential term represents a gaussian with a center of $t_o$, while the second exponential is a sinusoid with a frequency of $f_o$ and a phase shift of $\phi$.

The importance of Eq (2) is that it provides a mathematical model for signals which must be optimized in both the time and frequency domains simultaneously. However, Gabor's function was limited to one independent variable, time, and must be extended to two variables, $x$ and $y$, in the space domain before being applied as a model for simple cell response. The following section discusses that extension.

### *2.5 Extending Gabor's Function Into Two-Dimensions*

Daugman's paper on visual cortical filters (8) ties together important points regarding the visual cortex, filters, and the mathematics required in extending the uncertainty relation, Eq (1), into two-dimensions (2D). Based on the linearity of the simple cell response, and the localization of the cell's receptive field, Daugman stated the 2D uncertainty relation as (8:1162):

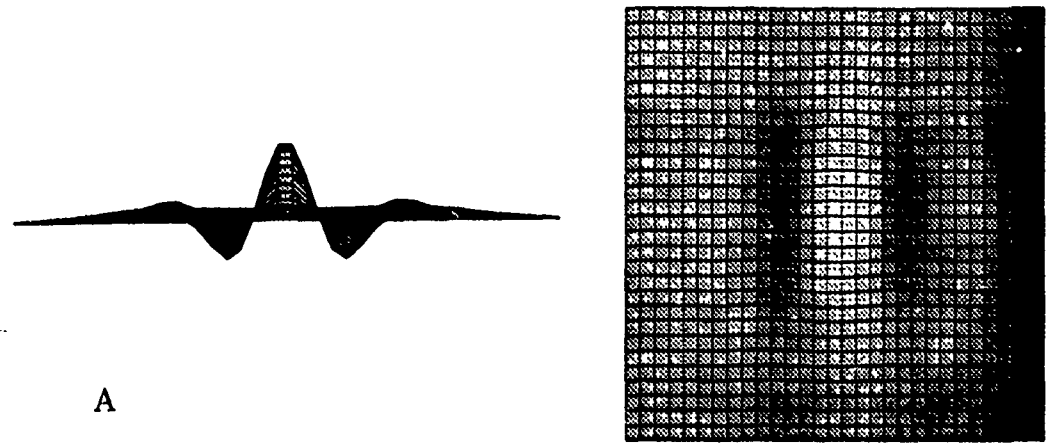
$$\Delta x \cdot \Delta y \cdot \Delta u \cdot \Delta v \geq \frac{1}{16\pi^2} \tag{3}$$

with $x$ and $y$ the prinicipal axes in the space domain, and $u$ and $v$ the principal axes in the frequency domain.

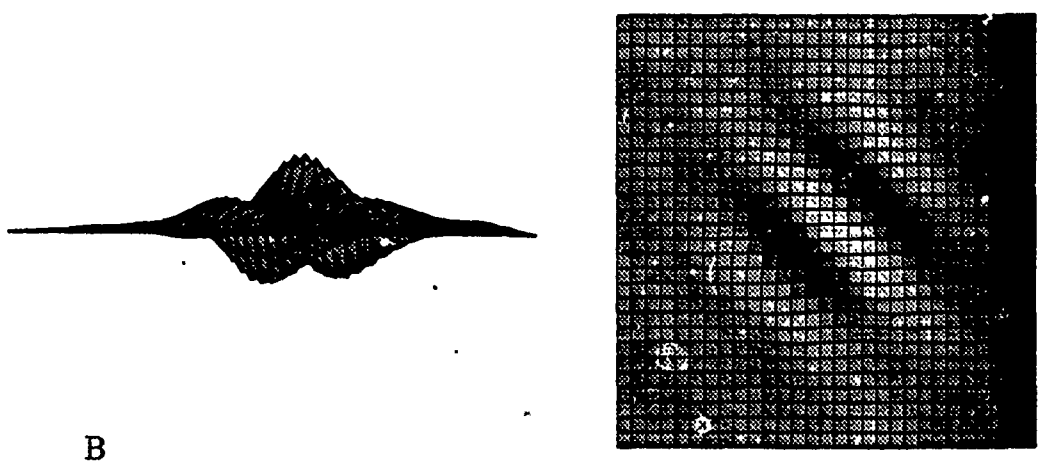
Daugman also extended Eq (2), Gabor's function, into 2D (8:1162). An equivalent form of this extension, taken from an article by Jones and Palmer (22:1236), is given here due to its similarity to Eq (2):

$$g(x,y) = K \cdot e^{-\frac{1}{2}\left(\frac{x_g^2}{a^2}+\frac{y_g^2}{b^2}\right)} \cdot \cos[-2\pi(U_o x + V_o y) - P] \tag{4}$$

where $x_g = x - x_o$ and $y_g = y - y_o$. The 2D Gabor function shown in Eq (4) is the product of a gaussian with an aspect ratio of $(b/a)$ and centroid at $(x_o, y_o)$ times the real portion



A



B

Figure 6. Examples of Two-Dimensional Gabor Functions

of a complex exponential having a spatial frequency of $\left(U_o^2 + V_o^2\right)^{1/2}$, an orientation of $\arctan(V_o/U_o)$, and a phase $P$.

Figure 6 provides examples of 2D Gabor functions. Figure 6-A shows a three-dimensional (3D) view of the real part of a 2D Gabor function having a unity aspect ratio ($b/a = 1$), centroid at $(0,0)$, and an orientation of zero degrees ($V_o = 0$), from viewpoints in the $x$-$y$ plane and along the $z$-axis. Figure 6-B is a look from the same viewpoints at a similar Gabor function, now with an orientation of 45° ($U_o = V_o$).

With a potential model now in place for simple cell response, the question turned to whether it was a correct model. The next section addresses that issue.

## *2.6 Re-evaluation of the Cat Visual Cortex*

A trilogy of articles by Jones and Palmer (23) (21) (22) tested the hypothesis that simple receptive fields in cat visual cortices are linear filters having the functional form of 2D Gabor filters. Besides this filtering model, Jones and Palmer felt that if this hypothesis proved true, a more important result might be the verification that the optimization principle, inherent in a Gabor function, would be applicable to other problems in visual information processing (22:1234).

*2.6.1 Spatial Structure Evaluation.* To measure the spatial receptive field structure of the cat's visual cortex, Jones and Palmer presented a random list of bright and dark stimuli to fourteen anesthetized cats. The stimuli consisted of small, rectangular grid sections (mostly 1 X 3, at times 1 X 5 and 1 X 1) flashed before the subject in some multiple of 5 ms (usually 50 or 100 ms). A nine-bit code was associated with each stimulus: four bits for the x spatial coordinate, four for the y coordinate, and one bit for the contrast (bright vs. dark)(23:1190-1193). This is shown in Figure 7 as the Stimulus List.

As these stimuli were presented within the cell's receptive field, every response (in the form of a spike) was recorded. The address in memory associated with the stimuli which forced a response was stored in the Spike Address List; the nine-bit code (described above) stored at that memory location was used to plot points within the appropriate correlogram (or, histogram). The researchers were able to obtain 55 spatial response profiles from 36 simple cells.

From their results they determined that the subregions within the receptive field which responded to either bright or dark stimuli, did not overlap to any great extent; there were no sharp transitions between these subregions or between subregions and the area surrounding the receptive field; and, the hypothesis that simple receptive fields could be modeled as either even symmetric or odd symmetric about a central axis was determined to be false in general. Lastly, it was determined that only half of the simple cells exhibited Cartesian separability. The issue of separability is an important one in determining how best to model the cell

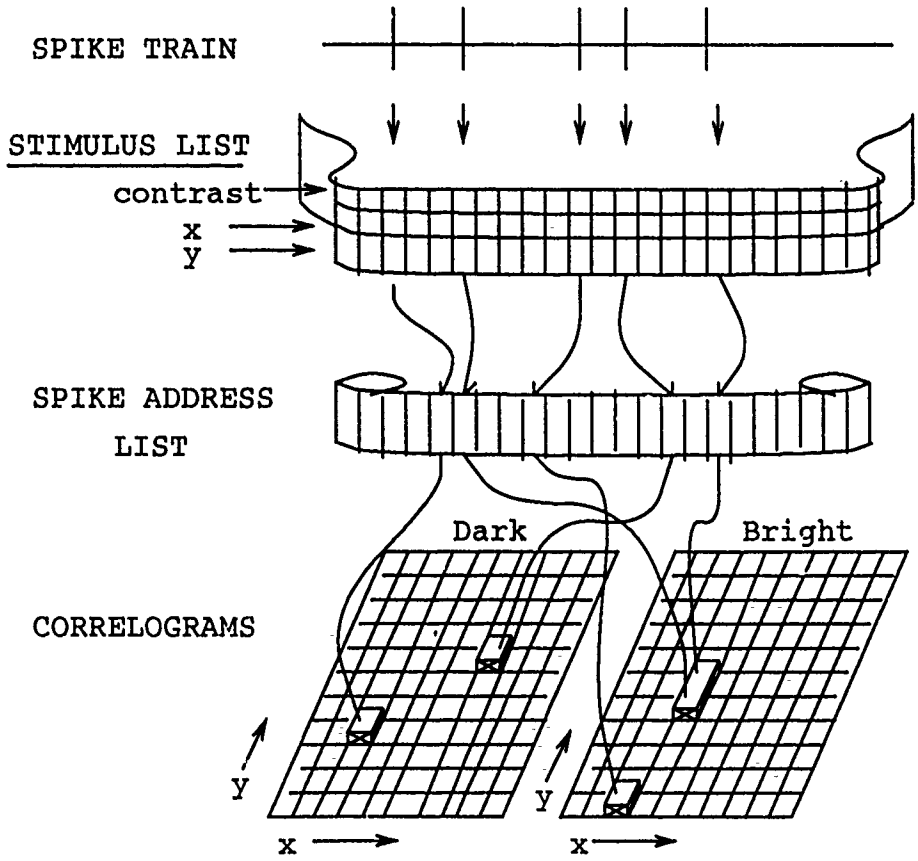



Figure 7. Cell Response Tracking Procedure Used By Jones and Palmer (23:1192)

response profiles. If Cartesian separability was the rule, then the 2D response profiles could be generated by two 1D profiles.

According to Jones and Palmer, Cartesian separability is exhibited when a receptive field can be stated mathematically as the product of a width axis function times a length axis function,

$$r(x,y) = w(x) \cdot l(y) \tag{5}$$

where $r(x,y)$ is the 2D spatial response profile, $w(x)$ is the width axis response profile, and $l(y)$ is the length axis response profile. For this separability to exist, various conditions must be met. Linear sections (*rows* of data) taken from the response profile parallel to the width (x) axis should be identical except for a scale factor. Likewise, linear sections (*columns* of

data) taken parallel to the length (y) axis should be identical to one another except for a scale factor. Lastly, the vector product of the average of the normalized sections should result in the 2D response profile. This was the procedure used in determining whether the 2D response profiles obtained were Cartesian separable; again, only half of the profiles could be considered as Cartesian separable.

However, every case where the separability failed a similar phenomenon was noted: "individual subregions were displaced from their expected positions along an axis parallel to their long dimensions.(23:1207)" , i.e., the subregions causing the profile to be non-separable need only be shifted along the profile's long axis to a point where separability may be achieved. While it was not obvious to the researchers what, if any, significance this phenomenon had in terms of neural image representation, they were able to summarize about certain restrictions which are placed on any potential model of the simple cell's receptive field, e.g., the model must allow subregions to have varying strengths (noncanonical phase), it must be smooth and continuous everywhere, and allow for both Cartesian separable and nonseparable forms (23:1208).

Once their evaluation of the spatial structure of the simple cell's receptive field was complete, the researchers turned their attention toward observation of the spectral nature of the receptive field.

*2.6.2 Spectral Structure Evaluation.* To determine the receptive field spectral response of the neuron, drifting sinusoidal gratings were presented to the subject cats, with the experimenter controlling the spatial frequency, orientation, and drift frequency of the grating. At spike occurrences, a symbol was added to a display at a location corresponding to the stimulus position in the 2D spatial frequency domain.

A contrast function response was then generated via selection of a specific spatial frequency, orientation, and temporal frequency. The percent of contrast was then randomly varied between eight levels (equipment restriction), and eight cyclegrams were generated (Figure 8-A (21:1215)). The cyclegrams shown demonstrate the average response after sev-

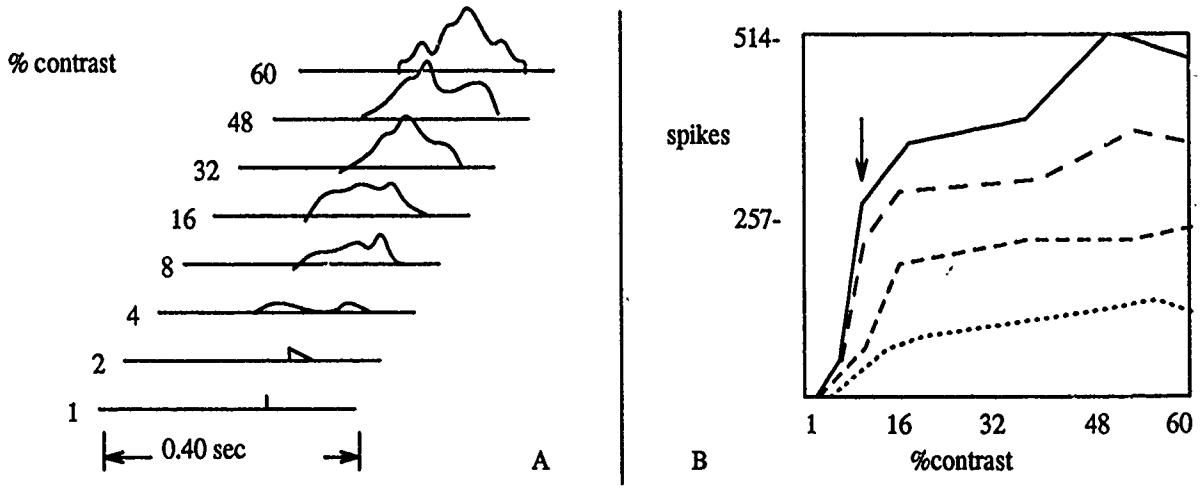



Figure 8. Non-Saturating Contrast Selection Process (21:1215)

eral cycles at each contrast. Figure 8-B shows the amplitudes of the DC (solid line), first (longer dashed line), second (shorter dashed line), and third (dotted line) harmonic components of each cyclegram. Their close parallelization to each other signified that the response waveform did not change as a function of contrast; as such, a non-saturating contrast was selected (Figure 8-B, arrow) for use in later experimentation. This non-saturating contrast would be used to distinguish between simple/complex cells as complex cells generally demonstrate an elevation in the DC term.

After a box was drawn around an area of responses, a grid was developed to represent 256 selected points in this region. The grid specified both the spatial frequencies and orien tations of the 256 drifted sinusoidal gratings. Five cycles of each grating (temporal frequency and contrast held constant) were presented, with responses being accumulated in the form of cyclegrams. Due to the noise inherent in the data being used, the surfaces of the resultant cyclegrams were smoothed with a 2D Gaussian filter. The output of the filter was labeled as the 2D spectral response profile (21:1216-1217).

The overall results showed the response to resemble a rectified sinusoidal modulation of the spike frequency. While the amount of rectification varied between cells, the form

was constant regardless of the spatial frequency, orientation, or contrast of the stimulus (21:1212). With these results in hand, Jones and Palmer would now evaluate the use of a 2D Gabor filter to model simple cell receptive fields in the cat striate cortex.

*2.6.3 Comparison to Gabor Filter Models.* Three criteria were mentioned as the deciding factors as to whether the researchers would accept the 2D Gabor filter as a receptive field model: simple cell 2D spatial response profiles and 2D Gabor filter spatial profiles should be indistinguishable; simple cell 2D spectral response profiles and 2D Gabor filter amplitude spectra should be indistinguishable; and, as the Gabor filter model presupposes linear spatial summation (see Section 2.3), simple receptive fields should also satisfy this constraint (22:1237).

*2.6.3.1 Spatial Comparison.* Jones and Palmer began by assuming that an observed profile, $r(x,y)$, would consist of a 2D Gabor filter, $g(x,y)$, a deterministic function of 2D space, $h(x,y)$, and additive Gaussian noise of zero mean and computational variance ${\sigma_o}^2$,

$$r(x,y) = g(x,y) + h(x,y) + e \sim N(0,{\sigma_o}^2). \tag{6}$$

For the Gabor filter to be a valid model, $h(x,y)$ will equal zero and the difference between the observed profile and the Gabor model will be distributed with zero mean and some variance (22:1241). That is, if $h(x,y) = 0$ we can expect

$$r(x,y) - g(x,y) = e \sim N(0,{\sigma_o}^2) \tag{7}$$

signifying that the Gabor model is valid as the only difference between the measured response and the Gabor is Gaussian noise.

First, 391 reverse correlations were performed, each with 256 samples, providing 100,096 samples of noise. The variance of the histogram's amplitude distribution was used as the variance of the noise. Secondly, a best-fit (least-squared error sense) Gabor filter model was chosen for comparison. Lastly, the variance of the residual error was compared with the

variance of the noise. In 33 of 36 cases, the Gabor filter satisfied the requirement that it account for all the variation in the data except that due to random error (22:1243).

*2.6.3.2 Spectral Comparison.* Two obstacles prevented the researchers from obtaining objective criterion in this portion of the evaluation. A single, expected error distribution could not be assigned to the entire data set due to cell response variability increasing with response amplitude of the histogram. Also, a 2D autocorrelation function could not be used to reveal any systematic 2D structure since the data were not collected with uniform sampling intervals (22:1243).

A best-fit Gabor filter model was again chosen by the least-squares error method. The residual error was calculated, and evaluated by eye. Two of the 36 cases demonstrated systematic deviation, while the remainder were considered excellent models of the response profiles (22:1245).

*2.6.3.3 Linearity Evaluation.* The evaluation was that if the 2D Gabor filter used to describe the spatial data and the 2D Gabor filter describing the frequency data have the same parameters, the receptive field combines spatially distributed inputs linearly. However, as with the spectral comparison, no specific statistical measurement was made, this time due to the lack of variance intervals about each estimated Gabor parameter. Rather, the linearity of the entire data set was evaluated through the use of scatter diagrams. No firm results could be made regarding the linearity of the Gabor models: "The 2D Gabor filter hypothesis rests firmly on the assumption of 2D spatial response linearity, but our evaluation of this conjecture produced mixed results (22:1253)."

*2.6.3.4 Conclusions.* Even with the lack of complete statistical measurement regarding both spectral and linearity evaluation, Jones and Palmer state that, "... the Gabor function provides a useful and reasonably accurate description of most spatial aspects of simple receptive fields" (22:1233); they conclude that the Gabor filter is a more than adequate model of simple cell activity. However, before the Gabor filter is labeled as *the* mathematical

model of simple cell response, some mention should be made that the description it provides is incomplete. The researchers point out that the issues of binocularity and time-dependent behavior of simple cells are not addressed (22:1257).

Now that a background has been provided for a probable model of cortical cell response, discussion of the study of visual anomalies is necessary. This will be covered in the following chapter.

# *III. Previous Evaluation of Visual Anomalies*

This chapter begins with a very brief history behind the study of visual anomalies and perception in general. A classification of these approaches, emphasizing the work Ginsburg has accomplished in one of these areas, is also provided.

## *3.1 Study of Visual Anomalies – A History*

Once upon a time (the late nineteenth century), Wilhelm Wundt led a drive which culminated in the separation of psychology from philosophy and physics, into its own branch of science. The major school of thought at that time was Structuralism, and its main tool was analytic introspection; consciousness could be considered the sum of many basic elements, including physical stimuli. As such, visual anomalies became a widespread topic of discussion and investigation, the idea being that if consciousness could be reduced to basic sensory elements, these anomalies would disappear (7:13,14). This relates very well to the use of Gabor filters to explain various visual anomalies, i.e., the combination (not necessarily summation) of many basic elements, in this case Gabor filters centered at various spatial frequencies, can provide insight/answers to certain visual anomalies.

Two new schools of thought were introduced in the early 1900's as the method of analytic introspection came under strong attack: Behaviorism and Gestalt psychology. The study of visual anomalies was virtually non-existent during this time due to the nature of these theories. Behaviorism effectively eliminated all mention of perception in its works; as the stimulus-response method was at the root of this theory, perception was disregarded due to the subjectivity inherent in trying to describe the conscious experience behind perception. Once the concept of perception was thrown by the wayside, visual anomalies were not far behind.

The Gestalt reasons for no longer studying visual anomalies were based more on theory than the Behaviorists' methodology. The Gestalt view of perception looked at the overall

picture – the entire 'form', for lack of a better word, of perception rather than studying basic units alone. There was no direct correspondence between the conscious representation and the physical stimuli; an anomaly was nothing more than another portion of normal perception(7:16-19).

About the middle of this century, both schools began modifying their positions, and, as a result, there was a resurgence in studying visual anomalies. Due to many occurrences where measured responses varied greatly with anticipated results, Behaviorists began using an intermediate, *nonobservable*, response known as the perceptual response. This created a grey area between response to a phenomena and a more subjective verbal answer to an internally perceived response. About the same time, Koehler and Wallach (27) helped change the direction of the Gestalt school through their physiological experiments regarding the modeling of electrochemical brain fields; visual anomalies were used to verify the researchers position that the conscious percept was similar in form to brain patterns(7:20). Various categories of perception research branched out from these rifts, and with them went approaches to investigating visual anomalies.

### *3.2 Categories of Perception Research*

While the types of theories which have arisen over the past fifty years is numerous (Direct Realism, Computational Vision, and Empiricism along with continuation of the Gestalt and Behaviorist theories (37:17,18)), they may be classified into two main categories: Cognitive, or Top-Down, theories and Bottom-Up approaches, based on neurophysiological approaches.

The Top-Down theorist bases perception on the higher cognitive level mentioned in the Introduction; the anomalies we see arise from previous perceptions. The Bottom-Up psychologist takes the view that the root of perception, and perhaps the reason for many visual anomalies to occur, lies in the stimuli processing which takes place prior to the cognitive level.

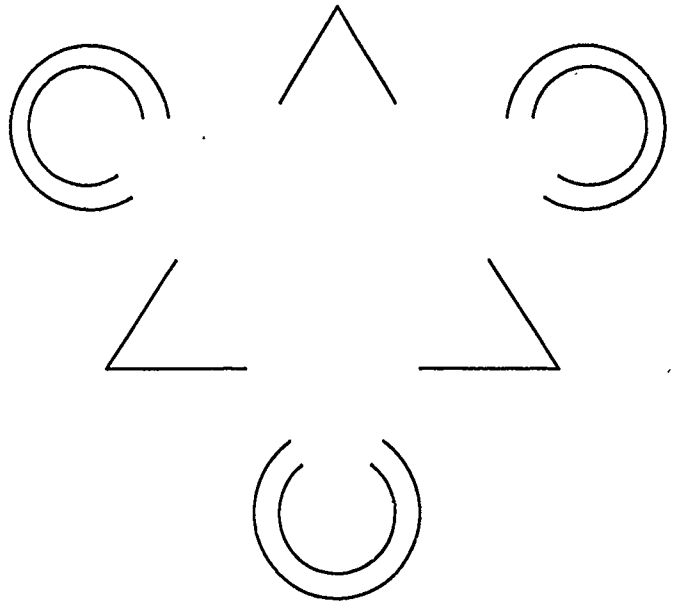


Figure 9. Kanizsa Triangle

As an example, how would each school view the Kanizsa triangle shown in Figure 9? Many, if not all, people will see (whether immediately, or after gazing at the figure for some length of time) a white triangle overlaid on three circular objects and a black-outlined triangle. However, what is actually drawn in Figure 9 is three 'pac-men' and three black-outlined angles – there is no white triangle placed atop various objects, yet this is what is most commonly 'seen'. Why?

The Top-Downer would be apt to say that the appearance of a white triangle covering three circles and a black triangle is due to the way we've observed things typically and have formed perceptions from them; the idea of three pac-men alongside three incomplete black triangles is so out of place with apriori knowledge that we try to associate it with something known, in this case the white triangle.

The Bottom-Upper has various tools at hand to evaluate this anomaly (the Boundary Contour System (15), and spatial filtering (14) approaches are two), but at the root is the idea that we perceive what we do because that is the way the higher levels are receiving data via the senses. Ginsburg has directed his efforts along this Bottom-Up path. His results will

be discussed in the following section.

## *3.3 Ginsburg's Study of Visual Anomalies*

*3.3.1 Background Leading to Ginsburg's Hypothesis.* Ginsburg initially studied geometric anomalies with the hope of explaining them in terms of spatial filtering (12:ix). By "explaining them" it is meant that researchers would have a better idea of how perceptual organization relates to the human visual system. As is appropriate, he begins by crediting psychologists for providing the greatest information regarding this perceptual organization. Through contributions in "psychophysics, experimental paradigms, and data analysis," (12:2) psychologists have advanced the knowledge of perception immensely. Ginsburg then suggests as a hypothesis:

> The major problem that the psychologists as well as the engineers have in pattern recognition research is that of defining a feature space in which like patterns or pattern elements cluster. Whereas the engineer is free to use any feature selection scheme that works, the psychologist is limited to metrics that must be correlated to human performance... It would appear that a valid model of the human visual system would be a logical source for developing metrics of visual form. (12:3)

Ginsburg based this hypothesis on suggestions that low-pass spatial filtering was this valid model (24) (25). If Ginsburg's hypothesis held true, both engineers and psychologists would benefit—engineers by having a precise functional model of the human visual system; psychologists by having quantified metrics of visual form (12:3). One notable success regarding low-pass spatial filtering is its use in distinguishing between handwritten letters (40). Ginsburg chose to evaluate visual anomalies, among other visual stimuli, since patterns which provide perceptual failure are as important to understanding the human visual system as are those which provide perceptual success (12:52).

*3.3.2 Application of Ginsburg's Hypothesis.* In his thesis, Ginsburg applied low-pass spatial filtering techniques to what might be considered "simple" geometric anomalies —

those types of anomalies assumed to be caused by proximity of lines or the angle between lines. The results were such that due to the filtering taking place along the neuronal pathway from stimulus input to visual cortex, it was very safe to assume that we perceive certain visual anomalies the way we do since we physically see them that way (12:52-56).

Ginsburg later expanded his research. Although he never explicitly stated a dependency upon spatial frequency information, he implied as much when stating the degree of how much spatial information is actually filtered by the retina: "...only a few ripples from the tidal wave of information that washes over the retina are needed to account for much of how we see objects" (13:10-11). The fact that spatial filtering is a frequency domain operation verifies this claim.

The method by which Ginsburg related low-pass spatial filtering to the activities of the human visual system was via a 2D filter based on, what was at that time, the most complete set of contrast sensitivity data available (5). This 2D filter, a modulation transfer function (MTF) shown in Figure 10 (13:136), is a logarithmic representation of a typical contrast sensitivity function. Figure 11 (13:138) provides a linear view of similar data and also demonstrates the two types of MTF used by Ginsburg. MTF(L) was derived by taking one quadrant of data available and applying symmetry to extend it to 2D. The MTF(H) was based on contrast sensitivity data from other studies (3) which emphasized the effects of contrast sensitivity at lower spatial frequencies. Figure 12 provides a 3D view of the MTF(H) (13:141).

Ginsburg extended his methodology to include band-pass filtering, and also included slightly more complex anomalies for evaluation (13:60-69,167-238). He summarized that indeed low-pass filtering was the major processing step taking place on input data... but not the only one. Ginsburg cited from Gregory that studies done on Zulu's, who do not see all of the geometric illusions Westerners do (12:64), suggested a selective attention mechanism. Ginsburg noted that this mechanism along with the pattern of eye scan and scaling effects are influences, but second and third order variables at best (13:67).

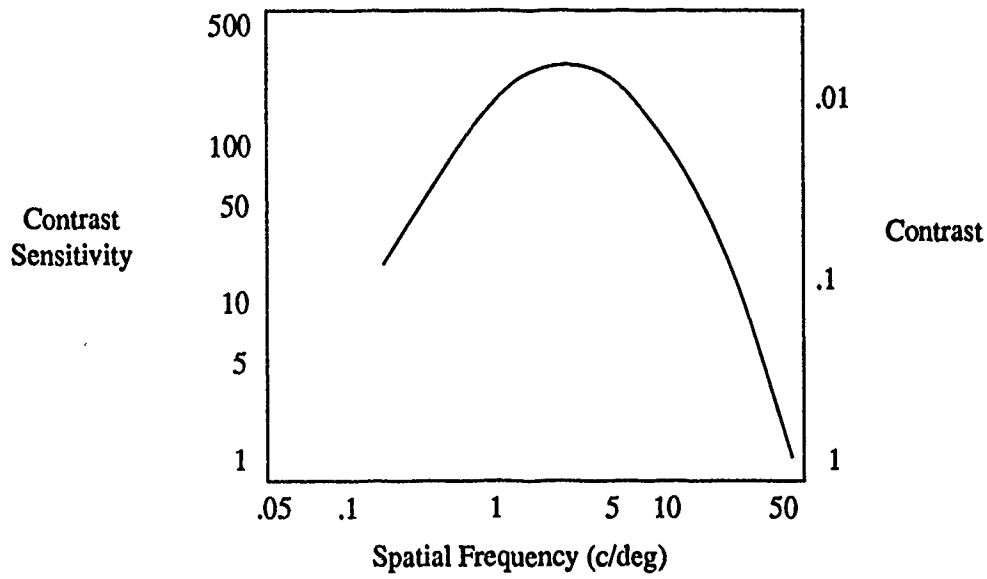



Figure 10. Typical Contrast Sensitivity Function (13:136)

### *3.4 Restatement of Problem Objective*

As another test of the appropriateness of using the Gabor filter as a model of simple cell receptive fields, anomalies originally investigated by Ginsburg via MTF's will be re-evaluated using the Gabor models. The following chapter will discuss the apparatus used to accomplish this task.

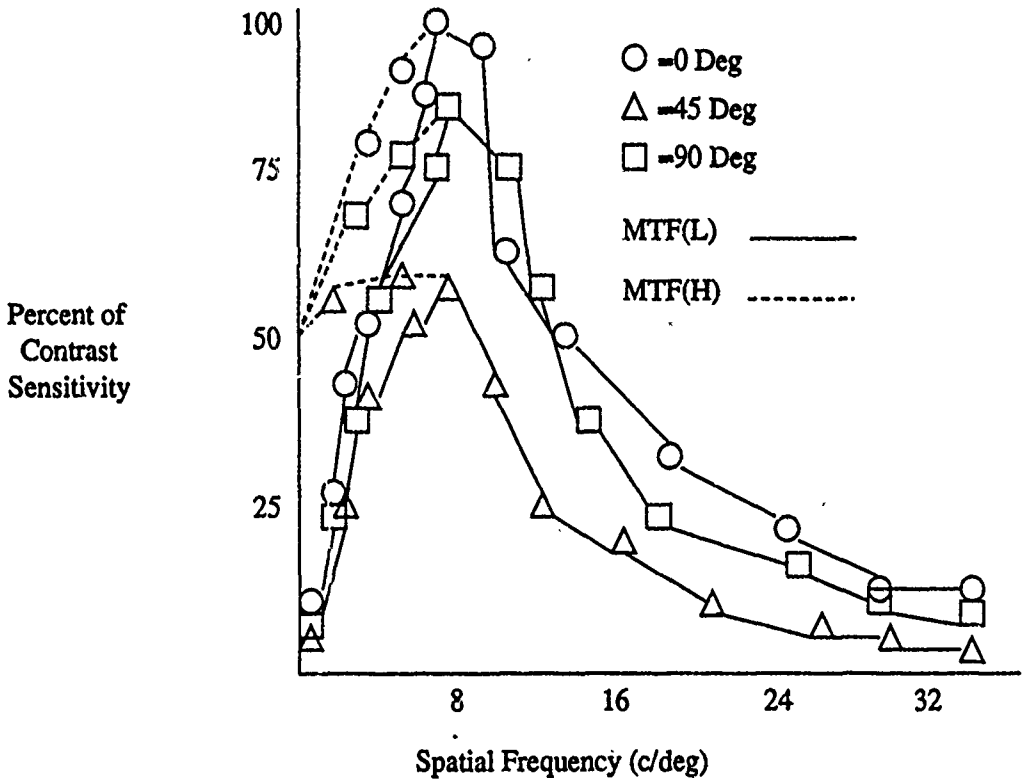



Figure 11. Normalized Contrast Sensitivity Functions for 0, 45 and 90 Degrees (13:138)

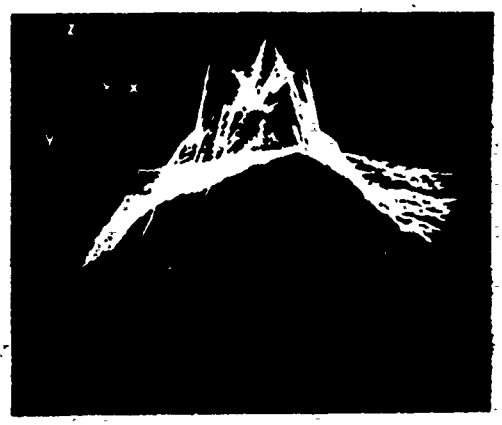


Figure 12. Three Dimensional View of a MTF(H) Spatial Filter (13:141)

# *IV. Methodology/Equipment*

This chapter will present the methods used to evaluate visual anomalies, along with a detailed description of the equipment used to accomplish this evaluation.

## *4.1 Methodology Overview*

Initially, the work performed by Ginsburg regarding his emulation of the MTF, particularly his evaluation of the Kanizsa triangle anomaly will be duplicated. Effectively, this emulation will be accomplished through the use of Gabor filters as will be further described. Following a satisfactory investigation of Ginsburg's Kanizsa triangle results, Gabor filter evaluation of additional anomalies will be investigated.

The method of anomaly evaluation will be as follows: an image file of the particular anomaly will be created via a paint program. A Fourier transform of this image will be accomplished, followed by filtering techniques determined by the particular evaluation taking place. An inverse Fourier transform will then be performed on the filtered image and displayed; it is this inverse Fourier transform which will then be evaluated regarding possible cause-and-effect. The above steps will be detailed in later sections.

## *4.2 Equipment*

The computer equipment used for this evaluation was a Sun Workstation$^{TM}$ (Sun-4$^{TM}$) running the SunOS Release 4.0$^{TM}$, along with a subcomponent of the Sun-4: the TAAC-1 Application Accelerator$^{TM}$ (TAAC-1). Computer programs/subroutines used for this effort were either originally written for use with the TAAC-1, or provided with the accelerator card by the manufacturer. This section provides an overview of the equipment used; Appendix A details TAAC-1 operation, specifically as it pertains to this effort.

*4.2.1 Sun-4 Workstation.* A typical Sun-4 consists of keyboard, mouse, and monitor. Two of the workstations on the Louvre file server, Dali and Monet, have an additional

external monitor utilized exclusively for displaying TAAC-1 output. The Sun-4 is a Scalable Processor Architecture (SPARC$^{TM}$), a type of Reduced Instruction Set (RISC) architecture, which emphasizes a simple instruction format, is register intensive, and executes instructions (with the exception of loads, stores, and floating-point operations) within a single cycle (45). The Sun-4 was chosen mainly for its TAAC-1 capability – although the inclusion of the external monitor makes programming tasks much easier as it allows a large additional area for display (in the case of Dali, 1024 x 1024 pixels) which may be viewed throughout any necessary program adjustments.

*4.2.2 TAAC-1 Application Accelerator.* The TAAC-1 is basically a computational engine connected to the Sun-4 via a bi-directional host bus interface. Its focus is on spatial and geometric data, intense computations, and displaying portions of its applications; as such, it does not perform any standard I/O routines and, as a slave processor, the TAAC-1 cannot write to the host. Hence, any type of programming requiring the use of both the TAAC-1 and I/O operations requires synchronization between host and TAAC-1 programs (47). This may be accomplished via a handshaking protocol which is detailed, along with other TAAC-1 specifics, in Appendix A of this document.

### *4.3 Specific Evaluation Procedures*

*4.3.1 Typical Image Characteristics.* The window size, into which the object under evaluation is placed, will be 128 X 128 pixels. In order to accurately adjust the filter's bandwidth in relation to object size, the object will also be sized, as near as possible, to 128 X 128 pixels. Any deviations from these standards will be noted in the appropriate discussion.

*4.3.2 Displaying Anomaly Images for Evaluation.* As previously mentioned, the method of evaluating anomalies begins with creating an image file of the anomaly using a paint program. A bitmap graphics editor, **touchup**, a public domain program currently found on Dali and Monet, was used for this effort. The output of **touchup** is a rasterfile; a rasterfile is

the Sun's file format for raster images. It consists of a header, a set of colormap values, and the actual pixel image. As the program used to display the image requires files in the Utah Run Length Encoded (rle) file format, a conversion is necessary on the **touchup** output. The raster-to-rle file conversion (**rastorle**) vertically inverts the desired image; a **rleflip** command is then used to re-invert the image back to the desired coordinates. The **gettaac** program can then be used to display the rle file on the TAAC-1 video terminal; simultaneously, the data which makes up the image is stored in TAAC-1 memory and is available for manipulation using the appropriate Fourier transform commands. These operations are summarized and detailed in Appendix B.

*4.3.3 Mathematical Operations on Anomaly Images.* Once the image data is stored in TAAC-1 memory, it can be manipulated via a host of different mathematical functions available in the TAAC-1 routine library (46). A typical process followed in evaluating anomalies consisted of first performing a 2D Fast Fourier Transform (2DFFT) on the image data, and storing the floating-point result in a separate location of TAAC-1 memory. This was accomplished using the supplied **t_fft2d** program. The filtering process of multiplying the 2DFFT result by some gain factor was done via user-generated programs, each distinguished by the type of filter being emulated – they are compiled, along with other software used, in Appendix C.

Appendix A does point out many of the subtleties of the TAAC-1; however, some mention should be made here regarding the programs written for the specific filtering modes required for this effort. While the algorithm that **t_fft2d** uses to accomplish the 2DFFT is fairly commonplace (decimation-in-time), the storage method for the floating-point output is not as common and should be well-understood before attempting to manipulate the data via non-library programs.

An inverse 2DFFT function, using **t_ifft2d** from the TAAC-1 library, was then applied to the filtered data, saving the result to yet another location in TAAC-1 memory. By storing the results from each step, each to separate memory locations, they could be viewed for

correctness and the data at that location could be accessed, when necessary, by using the TAAC-1 debugger, **tadeb.**

*4.3.4 Capturing/Storing of Video Data.* As results progressed to the point where images needed to be saved, either for future use or for printout, other Sun-4 commands became necessary. These are summarized in Appendix D.

Subsequent chapters detail results obtained via the methods explained in this chapter. Should any procedures deviate from this norm, they will be explained at that point.

# V. *Results*

This chapter will detail the results obtained via this research effort, beginning with the emulation of Ginsburg's work with the Kanizsa Triangle, in order to ensure a common baseline between the two efforts. The mode of filtering will then switch from the use of ideal LPFs to Gabor filtering. Lastly, other anomalies are presented which demonstrate both the appropriateness of using spatial filtering to emulate a portion of the human visual system, and the use of Gabor filters to accomplish this task. Even though some conclusions/evaluations are stated in this chapter, the scope of Chapter 5 remains the presentation of results; detailed evaluation/observation is presented in Chapter 6, *Conclusions/Recommendations.*

## *5.1 Definition of Terms*

Many terms used quite frequently in this chapter will now be defined for ease-of-reference.

- **Anomalous Contour** - A contour, not physically present, which is perceived in a figure. Referring to the picture of the Kanizsa Triangle, Figure 9, the anomalous contour is the triangle which appears to overlap the black objects. This term was chosen over other widely used phrases such as 'subjective contour' and 'illusory contour' (37:6) to emphasize that these contours have definite physical causes.
- **Cycles Per Object (c/o)** - In using various filters, a choice had to be made regarding the extent of the low pass filtering, i.e., how many harmonics of the Fourier Transform to keep for reconstruction of the original image. This choice was based on the number of sinusoidal cycles occurring over the breadth of the image, or, cycles per object. As mentioned in Chapter 4, images were created to encompass the entire window space to allow for easy computation of the c/o.

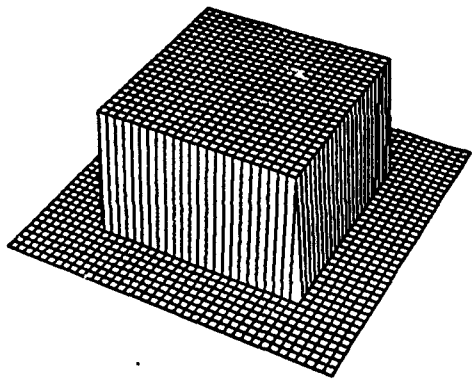


Figure 13. Example of Square Ideal Low Pass Filter

This parameter was chosen over the more commonly used cycles per unit of visual angle, or cycles per degree (c/d), as this study concentrated on possible static causes of anomalies, hence its being independent of viewing distance.

- **Square Ideal Low Pass Filter (SILPF)** - An ideal low pass filter will multiply the central portion of a magnitude spectrum by some gain factor (typically equal to 1), while multiplying the Fourier components outside of this area to zero. The term 'square' is included to refer to the shape of the base of the filter in the frequency plane. Figure 13 provides a 3D view of a SILPF.
- **Circular Ideal Low Pass Filter (CILPF)** - A CILPF performs a function similar to that of the SILPF, but has a circular base in the frequency plane as shown in Figure 14. It has been used mainly as a transition filter, in moving from the MTF-based SILPF to the Gabor low pass filter.
- **Gabor Low Pass Filter (GLPF)** - A generic term to describe one of many possible filters created via a combination of 2D Gabor functions. A detailed description of these filters is provided in Appendix E.

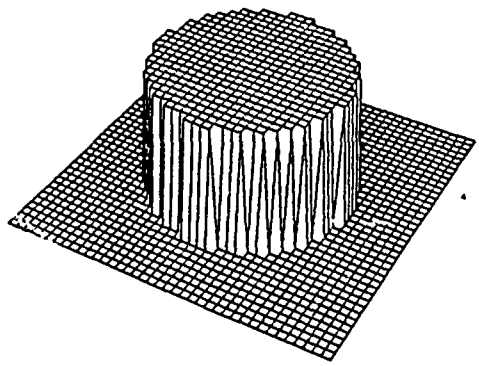


Figure 14. Example of Circular Ideal Low Pass Filter

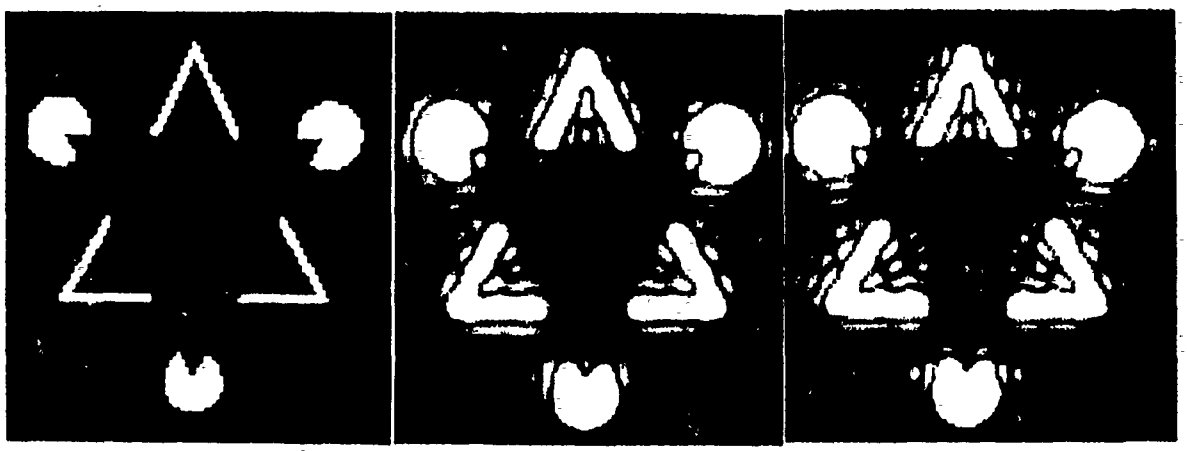



Figure 15. Results of Ginsburg's Filtering of the Kanizsa Triangle (13:225)

### *5.2 Emulation/Comparison to Ginsburg's Work*

Section 3.3.2 discussed the MTF used by Ginsburg for spatial filtering anomaly images. Besides this MTF filter, Ginsburg showed how a SILPF could be used to obtain results similar to the MTF. Figure 15 shows the original figure Ginsburg used for evaluation (A), the SILPF result (B - out to 16 harmonics), and the MTF(L) LPF result (C - again, out to 16 harmonics).

A point should be made regarding evaluation of Ginsburg's results as shown in Figure 15. Some critique of these results have been leveled regarding the existence of extraneous

A B C

Figure 16. Results of Using SILPF/CILPF on the Kanizsa Triangle

lines throughout the figures:

> ...when Ginsburg attempted to use a Fourier analysis procedure on a subjective contour-inducing configuration, the predictive ability of this technique clearly broke down. The analysis led to a complex lattice or net-like grid of line over the array. However, in such patterns one perceives only apparent contours, not a complicated interwoven pattern of lines. Thus it seems clear that any theory based on spatial frequency analysis is somewhat limited in its applicability to illusions. (7:216)

This critique has been chosen to emphasize a major obstacle teeming in the intellectual community, that of downplaying contributions made by members of the engineering disciplines. This most often arises due to a lack of understanding of the concepts presented and used to explain various aspects of anomaly evaluation. In this particular critique, the misunderstanding arises due to the idea that the 'lines' present 'draw' the outline of the anomalous contour triangle. They do not. It is the energy differences between the various areas ("physical intensity distribution" (13:65)) of the anomaly which suggest how data is being forwarded to the areas of the brain where concept formulation (i.e., object recognition) is taking place.

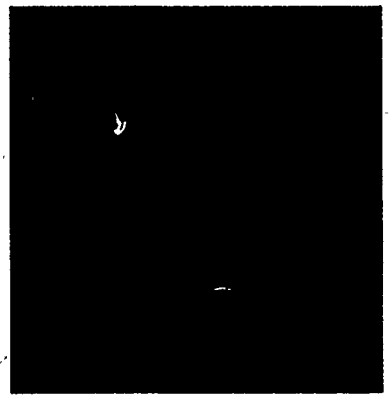


Figure 17. GLPF Result Using Summation of Four Gaussians

Figure 16 shows this effort's attempt at duplicating Ginsburg's results, part A being the original image, B the SILPF result, and C the CILPF result (both set to filter out everything above 16 c/o). The SILPF result closely imitates Ginsburg's; the CILPF result points out a slightly more pronounced intensity difference between the anomalous area and its surroundings. This output provided the confidence that the proper procedures were being followed through the filtering process; the next step was to use a GLPF on the original image and observe the results.

### *5.3 Use of Gabor Filter Version of the MTF*

Appendix E provides a version of the MTF(H) constructed with a summation of four Gaussians, one each at a radial distance away from the origin of 1, 2, 4 and 8 units, with the $\sigma$ of each Gaussian being $\frac{2}{3}$, $\frac{4}{3}$, $\frac{8}{3}$, and $\frac{16}{3}$ respectively (Figure 37). This configuration was chosen over a SILPF as it is a much closer approximation to the MTF which Ginsburg described. Figure 17 shows the output from the use of this filter; any indication of energy differences between the anomalous and surrounding areas has been removed. Could this result be the downfall of the Ginsburg hypothesis? Is spatial filtering not an accurate portrayal of what is happening in the human visual system? Will the Lions ever make it to the Super Bowl? The answer to all three questions is 'no'. (Sorry Detroit fans.)

In evaluating the differences between the results obtained in using the SILPF/CILPF

and the GLPF, it was noted that the 'lines' of energy visible in the SILPF/CILPF figures were indicative of the 'ringing' phenomenon associated with ideal filters. Note the sharp dropoff in the filter response for Figure 13 and Figure 14. Ideally, the change between the high and low response of the filter should be an instantaneous one, i.e., no time lapse between the two occurrences. In any physical system, including the human visual system, an instantaneous change is impossible. A side effect of using an ideal type filter is the ringing which takes place, manifesting itself in the form of the extraneous lines shown in the result figures. These lines are omitted in Figure 17 due to the slow, sloping dropoff demonstrated in Figure 37. Keeping in mind the nature of the simple cells and their spectral response, another attempt was made at creating a GLPF which might better exemplify what the human visual system is doing.

### *5.4 Use of Narrowly Tuned Gaussian GLPF on the Kanizsa Triangle*

Rather than limiting the GLPF to being composed of just a few Gaussians with relatively large spreads, a more correct approach would be to utilize many Gaussians, each having a smaller $\sigma$ than the previous Gaussians used (more narrowly tuned). This decision was based on the fact that each columnar grouping of simple cells has some sort of spatial/spatial frequency response aspect that they share; this might b the spatial location of the stimuli, or a particular frequency, where cell response is optimal. The typical spatial frequency bandwidth of cat simple cells is 1.32 octaves (8:1166). This was the value used in creating the filter shown in Figure 18.

In this figure, a narrowly-tuned Gaussian is placed at each harmonic up to a specific distance from the origin (in the following examples, out to 16 c/o), constituting a low-pass filter. The program **FineTunedGaussian.tc** (listed in Appendix C) was used in generating the result shown in Figure 19; the sharp cutoff inherent when using the ideal filters has been greatly reduced, yet the filter retains the quality of accurately emulating the desired biology. The result points out very strongly the energy differences between the anomalous triangle and its surrounding area.

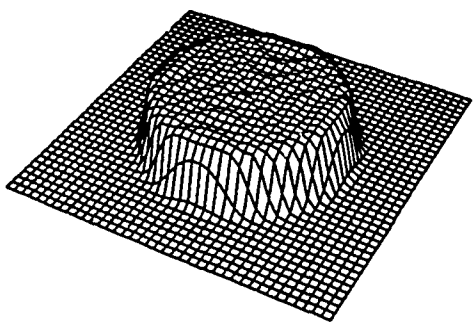


Figure 18. GLPF Created With Narrowly-Tuned Gaussians

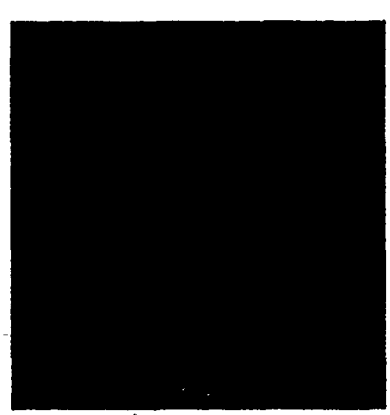


Figure 19. Result of Using Narrowly-Tuned Gaussian GLPF

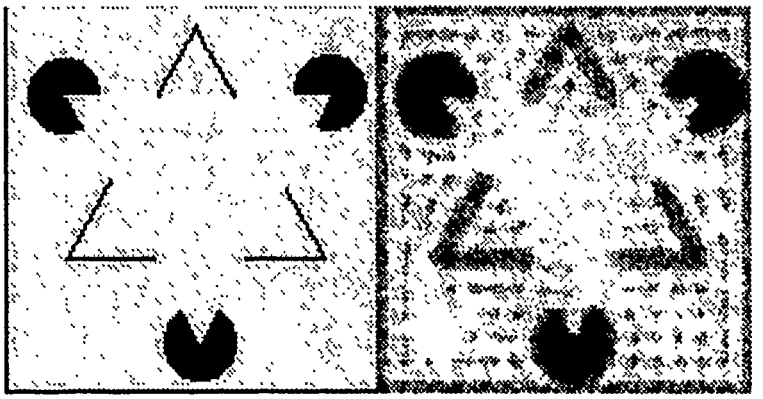


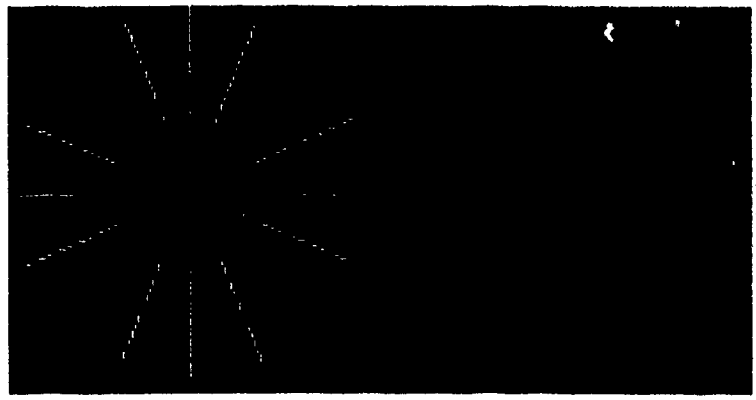

Figure 20. Result of Using Narrowly-Tuned Gaussian GLPF on Negative Image Kanizsa Triangle

To ensure that this result was not dependent on the contrast involved with the original image, an inverted form of the original image was created and the process rerun, again filtering out values over 16 c/o. The result is shown in Figure 20, with part A the original image, and part B the filtered result. Again, the energy differences are well-defined.

### *5.5 Gabor Filtering of Other Anomalies*

The narrowly-tuned GLPF was then applied to other anomalies, with a restriction on the anomalies being that some sort of anomalous contour should be involved. Low-pass filtering in general has already been proven to be a correct explanation for various line anomalies such as the Müller-Lyre and Poggendorf types. (Again, whether or not psychologists understand the physics behind the blurring associated with low-pass filtering, blurring which explains these line anomalies, is irrelevant.) Other types, such as the reversible Necker Cube, have their explanation in the user's focus of attention (what might be labeled as the Gestalt) rather than a direct application of filtering techniques.

*5.5.1 Gabor Filtering of the Spoked Circle.* The Spoked Circle anomaly, where the anomalous contour is a circle formed at the end of lines pointing inwards toward a center,



A B

Figure 21. Results Using a GLPF on the Spoked Circle Anomaly

as shown in Figure 21-A.

Figure 21-B provides the result of applying the GLPF process on the spoked circle, with Gaussians being used out to eight c/o. The proximity of the inner line-ends, relative to the outer ends, has allowed for a blurring to take place via the filtering process. A definite ring of energy is present where one perceives the anomalous contour.

Often associated with this anomaly is the perception that when a white background is used, the anomalous contour takes on a brighter appearance than its surrounding area (see Figure 22-A). Again, this phenomenon may be illustrated by using the GLPF process, the result of which is shown in Figure 22-B. There is a noticeable brightness in the anomalous contour area when compared to other areas in the figure. This is analogous to the Kanizsa results in Figure 20-B. Armed with this success, a similar type of anomaly was then evaluated – the Ehrenstein Illusion.

*5.5.2 Gabor Filtering of the Ehrenstein Illusion.* Ginsburg, in his doctoral work, presented a digitized version of the Ehrenstein Illusion where an anomalous contour is perceived in the area where horizontal and vertical lines approach one another but do not intersect (13:64;220). This anomaly is shown in Figure 23-A. As with the Spoked Circle, the GLPF is again used out to eight c/o with the result shown in Figure 23-B; as with the Spoked Circle,

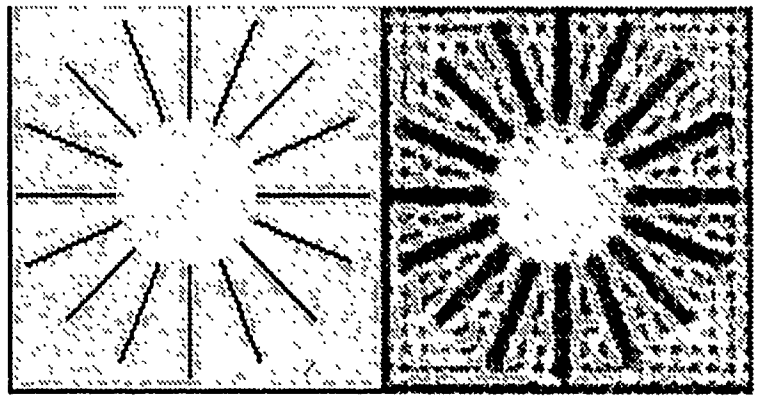



Figure 22. Results Using a GLPF on a Negative Image Spoked Circle Anomaly

there is a definite band of energy surrounding the anomalous area.

*5.5.3 Gabor Filtering of the 'Basic Block'.* At the root of the two previous anomalies is that which is presented in Figure 24-A, labeled here as the Basic Block. While it's use is obvious in the Ehrenstein Illusion, this is not the case in the Spoked Circle anomaly. The strength of the anomaly in the Spoked Circle lies in the number of spokes used; in viewing Figure 24-A, which may be considered the Spoked Circle anomaly with a number of the spokes removed, it is apparent that an anomaly remains – what needs to be determined is if applying a GLPF will still provide an explanation for the anomaly.

Using a GLPF out to 4.5 c/o provided the result shown in Figure 24-B. Once again, a definite energy ring surrounds the anomalous area.

The final chapter to this tome will provide a statement regarding what conclusions may be drawn from these results. Recommendations for future efforts in this field of endeavor will also be provided.

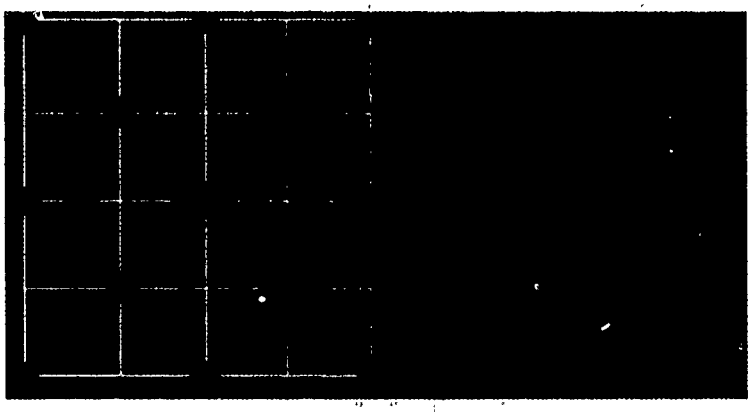



Figure 23. Results Using a GLPF on the Ehrenstein Illusion

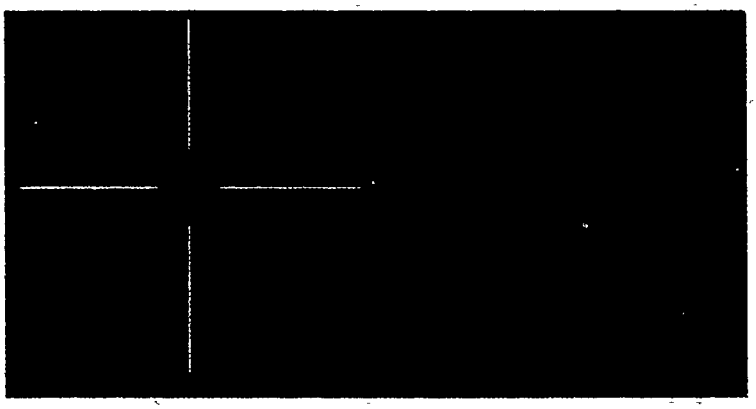



Figure 24. Results Using a GLPF on the Basic Block

# *VI. Conclusions/Recommendations*

This chapter will provide a discussion on the results compiled in Chapter 5. It will also detail paths of study that, due to time restraints, weren't studied in depth and should be considered in follow-up studies to this effort.

## *6.1 Conclusions*

*6.1.1 Decisions Leading to Final GLPF.* The results provided in Chapter 5, especially those related to the narrowly-tuned GLPF, are quite exciting and, admittedly, surprising. Surprising because it was fully expected that the initial GLPF used (Figure 37) would show the type of energy differential which would strongly suggest that the correct emulation of the biology was taking place. Exciting because once the correct GLPF was determined, the results were more convincing than expected.

Initially, the Kanizsa Triangle was chosen to be the first anomaly investigated not only because of Ginsburg's work with it, but also due to the much more complex nature of the anomaly when compared to others where anomalous contours are not present. By explaining the anomaly via a known model of simple cell response on the visual cortex, a much larger subset of anomalies could then also be explained.

The originally disheartening results shown in Figure 17, where the MTF was emulated with variably spaced Gaussians, led to two possible conclusions: spatial filtering was not an appropriate metric in the evaluation of these anomalies, or, the combination of Gaussians comprising the GLPF was incorrect. The second case was investigated as the likelihood of error was higher here, i.e., spatial filtering was used often enough in the past as a feature detector ((26), (4)) to doubt its misuse. A step back had to be taken for a review of what is currently known about the biological activities taking place from retina to visual cortex.

It is known that within the visual cortex there are columnar grouping of cells, each column applicable to a specific characteristic of the input stimuli, e.g., spatial frequency,

right vs. left indications, color ('RGB detection'), and motion detection ('Y' cells) (42). As mentioned in Chapter 3, based on the amount of information available to the retina versus the amount of information passed from the retina, some reduction in data (filtering activity) is taking place – from the results gained in using low-pass filtering for segmentation (blob detection), this was the primary activity to emulate. How then to best combine the Gabor filters within this low-pass scheme, if the original MTF configuration was incorrect?

In retrospect, the elimination of many harmonics within the first GLPF Gaussian-combination seemed out of place with what would actually be happening on the visual cortex. Intuitively, it seemed that all harmonics within the limit of the low-pass filter would contribute to the final outcome; based on previous work by Daugman (8), these contributions should be narrowly-tuned Gaussians. These factors combined into the final GLPF shown in Figure 18 – one that integrates the spectral response of many simple cells into what may be considered a columnar response of cells respondant to the spatial frequency of input stimuli.

*6.1.2 Gabor Filtering's Role in the Visual Process.* Ginsburg's doctoral dissertation includes a detailed discussion on spatial filtering's place in the realm of object recognition (13:93-97). The main emphasis in that writing, as will be here, is that filtering (in this case with Gabor filters) is but one small portion of the processing taking place prior to a final perceived model of the outside world being passed to the concept forming areas of the brain.

As mentioned previously, certain groupings of simple cells, arranged in columnar form, respond in like manner to specific input stimuli. The complexity of the human visual system lies in the variances that these cell columns will respond to. Color, motion and, perhaps most importantly, the temporal nature of the input, are factors that must be accounted for when discussing the human visual system in total – this effort has concentrated on only one of these areas, spatial/Gabor filtering.

The strongest critique, or at least most common, of spatial filtering's role in the visual process, stresses the views of Prazdny (38) and Shapely and Gordon (44) as proof that linear spatial filtering is an incorrect approach to analyzing anomalous contours. These

arguments are based on the spatial separation and contrast differences of inducing elements. The rebuttal to these arguments consists of many factors; Ginsburg, as one of a handful of engineers amidst a gaggle of psychologists investigating anomalous contours, lists many (37). The bottom line is that most of the documentation on anomalous contours is written by psychologists who either do not know enough about linear spatial filtering to make qualitative remarks about it, or, misinterpret the claims that linear spatial filtering does provide an explanation of anomalous contours, *albeit a restrictive explanation regarding only one small portion of the visual process.*

The following section will relate the Gabor filtering process with other areas of the human visual system – ones that should be investigated by future efforts.

*6.1.3 The Kanizsa Triangle.* Figures 19 and 20 emphasize the fact that spatial filtering plays an extremely important part in understanding the causes of perceiving anomalous contours. As the filtering process is applied to the original image, the blurring associated with the low-pass filter causes a spread of energy outward from the pac-men and angled lines. This blurring is restricted to areas outside of the anomalous contour. In the black-background image (Figure 19), the blurring is from the higher energy white objects; a lower energy region encompasses the original anomalous contour. The opposite was true with the white-background Figure 20-B – the anomalous contour region had a higher energy content than the immediately surrounding area.

It was noted that the differing energy band surrounding the anomalous area was finite, i.e., it extended away from the anomalous area not much farther than the pac-men and lines did. To test whether these brightness differences are also perceived as finite (and if so, the extent), an extremely informal survey was designed with the following approach: a 256 X 256 pixel version of Figure 15 would be placed in the middle of the 1024 X 1024 pixel TAAC-1 screen, with all pixel values, other than the Kanizsa Triangle image, set to black. Individuals are then asked if an anomalous contour triangle could be seen; those answering positively would then be asked if the anomalous area appeared darker than its surroundings;

finally, those answering positively are then asked to point to an area of the screen where, if at all, the background had the same brightness as the anomalous area, i.e., the extent of the energy difference from the darkness of the anomalous area. The hypothesis is that, rather than the anomalous area being brighter, there is a higher energy ('whiter') band around the anomalous contour corresponding to the higher energy displayed in the spatially filtered result. A recommendation to accomplish this is provided in Section 6.2.1.

However, on their own merit, the Kanizsa Triangle results are quite remarkable. By applying a more physiologically appropriate filter than those previously used, there can be no question that low-pass spatial filtering is one of the causes of anomalous contours being perceived.

*6.1.4 The Spoked Circle/Ehrenstein Illusion/Basic Block.* These types of anomalies were perhaps first noted by Ehrenstein (10). It is within his descriptions of anomalous circles that another discrepancy was noted in how many psychologists view this type of illusion: "...the increase in brightness of the central area is greater ...", "...the increase of induced brightness within the central white area". What has been labeled in the past as 'induced brightness' is actually the result of the 'energy smearing', a.k.a. blurring, taking place in the area immediately surrounding the anomalous contour. Again, this blurring is the effect ...spatial filtering is the cause.

The blurring shown in Figures 21 through 24 is, in retrospect, perhaps not extreme enough to point out the energy difference between the anomalous region and its surroundings. While there is certainly an area of enclosure around the anomalous regions, only in Figure 22 is the energy spread within the 'spoked doughnut' well-pronounced.

The highlight of the results regarding these three anomalies, was the ability to progress from the Spoked Circle, where many lines surrounded the anomalous circle, to the Ehrenstein Illusion, where the fewer lines outline the anomalous figure, and finally to the Basic Block, where the anomalous area is outlined with only a minimum number of spokes. It is noted here that while the anomalous contour in the Basic Block, Figure 24-A, is certainly not of

the shape shown in the spatially filtered result, that discrepancy is due to the original image not being centered in the 128 X 128 pixel area.

## *6.2 Recommendations*

*6.2.1 Individual Observations of Kanizsa Triangle.* As mentioned previously, some sort of method could be developed to question individuals as to the extent of their observing a ring of brightness around the anomalous triangle. In general, this should correspond with the extent of blurring taking place when the GLPF is applied to the original image.

There are several pitfalls to overcome if this approach is to be tried. The glare off of the monitors is noticeable enough to conflict with the desired all-black background necessary to approach valid results. If the monitor could be moved to a light-free area, such as the back corner of Room 2011, Building 642, where drapes are available to block out extraneous light sources, then more confidence could be placed in the individual responses. Another alternative might be to take a non-glossy, black and white photograph of the triangle while displayed on the TAAC-1 for use as the input stimuli.

But perhaps the most difficult thing to overcome with this type of approach is the subjectiveness inherent in this type of survey. Responses from individuals, except those responses which are automatic in nature, are always open to skepticism – either as a result from suggestive questions imposed, the inclusion of the concept forming areas of the brain, or personality traits. Still, as a very informal method of giving some credence (read 'credence' as 'warm fuzzy') to people perceiving a blurring of energy around the anomalous area, this approach might suffice.

*6.2.2 Mapping of Input Data Onto the Visual Cortex.* It is known that within the data path from retina to visual cortex, a coordinate transform takes place moving from the Cartesian coordinate system to a form of the Polar coordinate system (34). In what has been labeled as the LogZ mapping, the x-y plane associated with the Cartesian system, is replaced with a log z-$\theta$ plane. This coordinate transform allows for scale invariance (transforms to a

shifting along the log z axis) and rotation invariance (a shifting along the $\theta$ axis) regarding input data. Some concern is raised that should this LogZ mapping occur prior to the spatial filtering process taking place, the effect of blurring would not carry over into this new domain. One factor suggests that this concern is negligible.

It was noted that the meat does not do an exact replica of a LogZ transform, i.e., due to the physical structure of the visual cortex there is a slight bending inwards of the mapping. A spatially filtered version of this curved mapping would still have a pronounced energy difference between the anomalous area and its surroundings; e.g., the ring of energy which surrounds the anomalous area in the Spoked Circle would still be present even if spatial filtering followed the LogZ mapping (42).

Some facilities are set up at AFIT to perform this LogZ transform. It is suggested that both approaches be taken regarding evaluation of spatially filtered images. First, take the spatially filtered results from this effort and apply the LogZ transform to them, observing if the energy difference between areas still applies. Secondly, transform the original, non-filtered, images to LogZ, filter this image with the GLPF and also observe if the result demonstrates energy variances strong enough to verify the claims made in this report.

*6.2.3 Temporal Aspects of the Visual Process - Part I.* It was previously mentioned that perhaps the most important trait of data being input to the human visual system are its temporal characteristics. Most commonly associated with the temporal aspects of input data is the length of time it is presented in one's field of vision. Will the strength of the anomaly increase or diminish as time increases? Will the claims made in this effort regarding anomalous contours being created as a result of Gabor filtering hold up under restrictive image presentation?

If one could absolutely fixate a subject's focus at a specific point and use an instrument, such as a tachistoscope, to flash an image before that subject, the minimum length of time necessary for the subject to recognize the anomaly could be recorded. Two drawbacks for accomplishing this exist at the moment: one, unless the image is presented very briefly, the

Gestalt of the scene would vary each time the image is flashed due to eye saccades. This puts a damper on the results obtained from performing such an experiment. The second drawback is the lack of an accurate tachistoscope for use in the AFIT Signal Processing Lab. Using batch files and the **taread/taload** commands detailed in Appendix D, a very simplistic version of a tachistoscope was created. Basically, an image is loaded to the TAAC-1 screen; this is followed by the image being written over by a black background. To make this set-up worthwhile, other issues, such as the time to execute each command and the proper amount of delay to add between the two screen writings, must be investigated.

*6.2.4 Temporal Aspects of the Visual Process - Part II.* While the inclusion of the lower harmonics allows for blob detection, the passing of the higher frequencies will accomplish edge detection. The results shown in Chapter 5 all relate to low-pass filtering only, yet it is obvious that the visual process is also accomplishing high-pass frequency computation based upon the ultimate sharpness of the perceived image. Note the use of the word 'ultimate' in this case is to suggest a time dependency, i.e., we ultimately perceive an image, in the case of the Kanizsa Triangle, which has sharply defined lines and pac-men. It is conjectured here that as more time is available to observe an image, the spatial frequencies being computed moves from the lower harmonics out to the higher.

For example, the results provided in Chapter 5 would be the result of an initial scan of the anomaly; using the tachistoscope suggested in the previous section, it is hypothesized that as the image is presented for shorter lengths of time, the responses will be of perceiving blurrier images. Alternately, as the image is presented for a longer time span, the sharpness of the image will be noted due to the inclusion of the higher frequencies. Similar to the balance which exists between the spatial/spatial frequency processing going on in the human visual system (as evidenced by the applicability of the Gabor model of simple cell response), this hypothesis suggests a synergy between the temporal and spatial frequency characteristics of input stimuli.

Perhaps the most important aspect of this temporal processing goes beyond the human

visual system to encompass all processing taking place amidst the meat between our ears. This hypothesis of temporal/spatial frequency interaction might also extend into the realm of consciousness.

No, there is no claim made here of any expertise in the area of philosophy, which seems to be a prerequisite in any discussion on the feasibility of teaching computers to think (43) (6). No, there are no hardfast methods presented here to back up the claims made ... this is pure conjecture, based on some knowledge of processing taking place in the brain. (Remember the phrase: "A little knowledge *could* be dangerous ... but it's enough to make you a psychologist.") There is some finite time associated from presentation of data before the eye to brain activity indicating a registration of that data; that time is generally considered to be in the range of 100-200 milliseconds (41). Rather than a comparison of exact times though, this discussion will relate processing times in terms of generic units. When data is presented before the eye, as stated, there is a finite amount of time before a concept is formed and cognition of the object or scene takes place. What goes on during that time frame? The image is mapped onto the visual cortex, memory is accessed to match this image with known data, and a concept is formed based on the object or scene presented.

This is not to suggest a purely serial process taking place. It is known that the brain does not process data in a serial manner, nor even in a parallel manner (associated with today's concurrent processors), but rather in a neural computation mode where layers of processing take place. The point to be made is that, regardless of the brain's processing mode, an external object must be perceived before it may be conceptualized. This is not true regarding internally generated, or self-induced, concepts. The thoughts that we have are concepts formed at an unimaginable speed due to almost instantaneous access of previous concepts stored in memory. The claim here is that consciousness, or self-awareness, has as its root the time difference between formalizing concepts as a result of external stimuli vs. self-inducement of conceptualization.

At first glance, this appears to be a recursive sort of definition – that self-awareness is due to the self instigating some process – but this is not the case. It might be better to define

self as it is used here as the process (however that process may occur) of automatic memory access without inducement from outside sources. Self-awareness may then be defined as the ability to distinguish between concepts formed from external stimuli and those formed from self-inducement; one metric for the self to use in this awareness process is that of time.

Our knowledge of these differing time bases may lead to another sort of anomaly, that which is often labeled as 'deja vu', or an intuition or feeling that one has experienced something they are presently observing, or going through, at a previous time. This anomaly might simply arise from having glanced at a scene or object for a brief amount of time; an amount of time which the self does not register as typical for either the self or external input. Ergo, the uncertainty by the self of whether this image has been seen before or not.

It is acknowledged that this hypothesis of the conscious self being nothing more than a time differentiator is speculation at best based on what little is known regarding the concept forming areas of the brain. It *is* mentioned, however, to emphasize the importance of investigating anomalies at all levels of processing – especially if an accurate, dependable machine is to be developed which can emulate the human visual system.

# Appendix A. *TAAC-1 Application Accelerator*

Although not all of the tools available for use with the TAAC card were utilized for this effort, those that were are documented here with a brief description of their use with hopes that future efforts using the TAAC-1 card will be made much easier.

## *A.1 Hardware*

The overall goal of the TAAC-1 is to provide the processing speed of dedicated, special purpose hardware while providing the user with a highly programmable, easy to use, tool. Speed improvements over microprocessor assembly instructions have been achieved by using the parallelism inherent in many algorithms to accomplish multiple tasks within a single processor instruction. Greater detail of each of the following subsections may be found in the TAAC-1 User Guide (47).

*A.1.1 Data/Image Memory.* TAAC-1 memory is dual-purpose – it may be used as storage for large data sets or as a frame buffer to store images for display. While the hardware used to fabricate the dynamic random access memory (DRAM) is important, the discussion here will concentrate on the 1D/2D addressing scheme of the DRAM, shown in Figure 25. 8Mbytes of memory are available, set up as 2M 32-bit words, with the basic memory structure being a *sector* consisting of 1Mbyte of memory. Some important points to note regarding Figure 25:

- Many of the TAAC-1 library programs utilize the 2D addressing scheme, shown outside the blocks.
- However, many of the tools available to view the actual data, vice the image appearing on the screen, require a knowledge of the 1D addressing scheme, shown inside the blocks.

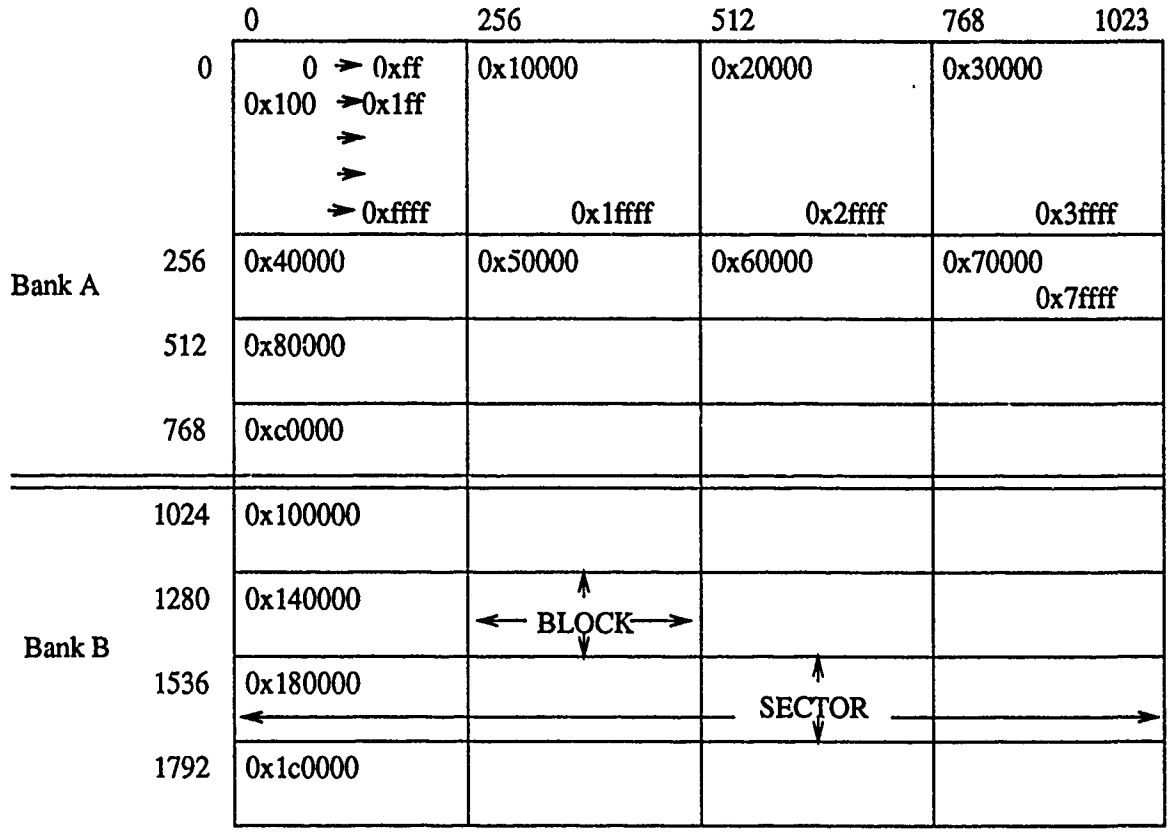



Figure 25. Memory Map of the TAAC-1 DRAM. 1D addressing within the boxes; 2D outside the boxes (47:2-6)

- Both Banks A and B are available for data storage, although displaying data is not equivalent between the two banks. The video monitor associated with Sun-4 Workstation Dali is 1024 X 1024 pixels; as such, only Bank A data will be displayed. Similarly, the 512 X 512 monitor alongside Sun-4 Workstation Monet will display 2D addresses 0,0 through 511,511 of Bank A only.

*A.1.1.1 Linear Addressing.* In the 1D addressing mode, a sector consists of 4 X 256 X 256 X 32 bits of memory. Figure 26 shows how 32 memory chips addressed in parallel can access four 32-bit words from an 8-bit row address and an 8-bit column address. The 1D address map is shown in Figure 27.

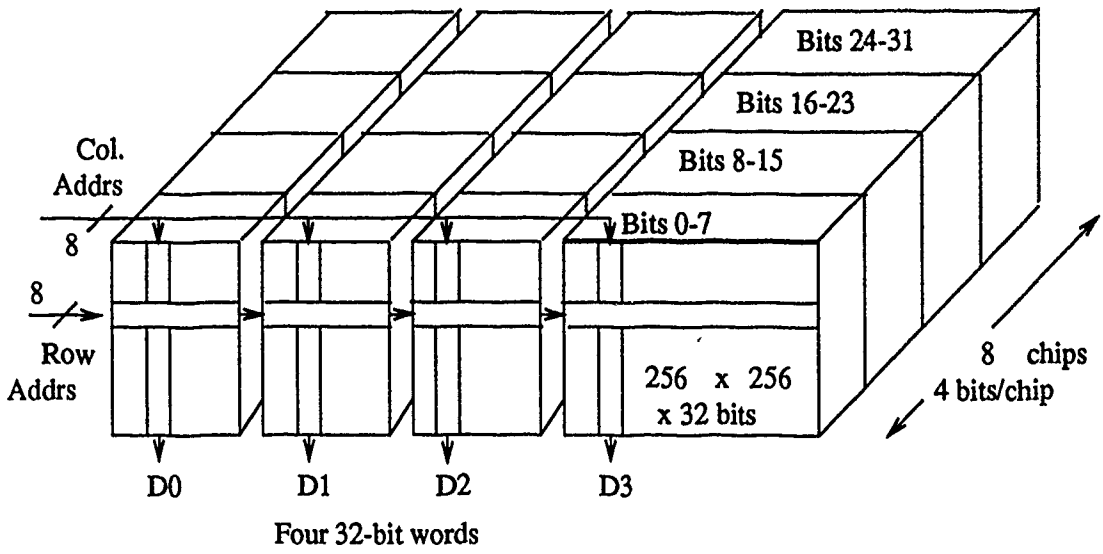



Figure 26. Sector Organization of the TAAC-1 DRAM (47:2-20)

*A.1.1.2 Image Addressing.* In 2D space, the 8Mbytes of memory are organized as a 1024 X 2048 X 32-bit image array, with 10 bits of spatial $x$ addressing and 11 bits of spatial $y$ addressing. The three high-order bits of the $y$ address signify the sector, while the two high-order $x$ bits distinguish the block within the sector (see Figure 25 for representations of blocks and sectors). The bit shuffling which is shown in Figure 28 is accomplished in hardware; however, it is important to note how the TAAC-1 expects to receive this 2D address for image manipulation.

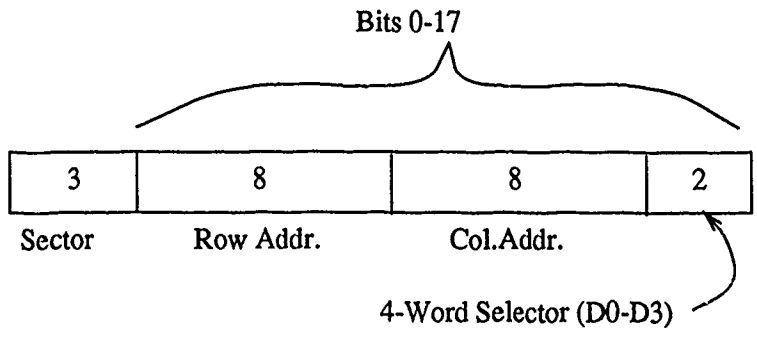



Figure 27. One-Dimensional Address Map of the TAAC-1 DRAM (47:2-21)

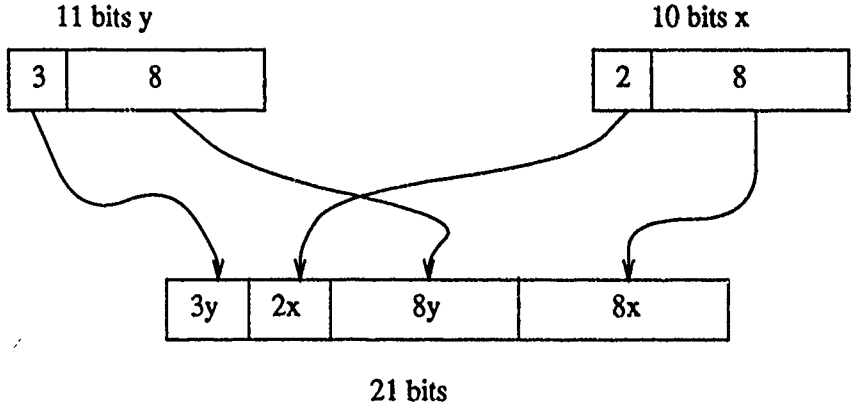



Figure 28. 2D to 1D Address Transform of the TAAC-1 DRAM (47:2-21)

An example will best describe this process. If the data to be accessed is at 2D location X = 512, Y = 768, the following line of C code could be used to inform the TAAC-1 of the source image:

$$\text{source} = (\ (Y << 16)\ |\ X);$$

That is, left shift the spatial $y$ address 16 bits then bitwise OR that result with the spatial $x$ address. This places the $y$ address in the high-order 16 bits of the 32-bit word, and the $x$ address in the low-order 16 bits. Again, the TAAC-1 hardware will handle the bit shuffling shown in Figure 28.

*A.1.2 Video Output.* The 32-bit data output to the video monitors serving the TAAC-1 is separated into four 8-bit channels as shown in Figure 29. Each of the 8-bit color channels of video (red, green and blue) passes through its own video lookup table and digital-to-analog converter (DAC), providing $2^{24}$ possible color blends. The alpha channel (most significant bits 24-31) may be used as an 8-bit overlay channel if the appropriate overlay masking is used. This allows 256 possible overlay colors to be selected out of the $2^{24}$ available.

These color representations became important during this effort regarding the display of data after each portion of the anomaly analysis. For example, the 2DFFT TAAC-1 library program, **t_fft2d**, accepts integer input and outputs data in floating-point form. The

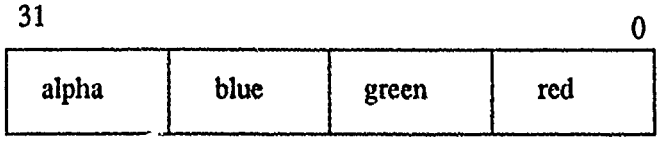



Figure 29. Eight-Bit Color Channels of the TAAC-1 DRAM (47:3-11)

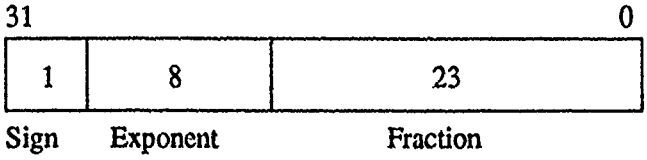



Figure 30. Single Floating-Point Format, IEEE Standard

TAAC-1 uses the IEEE standard for single floating-point representation, as described by the following equation (48:57):

$$value = (-1)^S * 2^{E-127} * (1.F) \tag{8}$$

where S is the sign bit, E is an 8-bit exponent in modulo-127 notation, and F is a 23-bit fraction in sign-magnitude form. S, E, and F are shown in 32-bit form in Figure 30.

In comparing Figures 29 and 30, it is obvious that a color display representation of floating point data output is meaningless. When the TAAC-1 output *is* meaningful regarding video display, as is the case in the integer result of the inverse-2DFFT routine (**t_ifft2d**), a brighter pixel intensity does not necessarily mean a larger integer value. As an example, $100_{16}$ is displayed as an extemely dark shade of green, while $0FF_{16}$, a lesser numerical value, shows up as a very bright shade of red. A short TAAC-1 program (**colorblend.tc**) was written for quick display of color ranges; the code for this program is included in Appendix C.

*A.2 Utilities*

Several TAAC-1 utilities are available to be run directly from the Sun OS prompt. Those which were used for this effort are described in the following paragraphs.

*A.2.1* **tainit.** This command will initialize the TAAC-1, and *must* be used prior to the running of any programs which utilize the TAAC-1 DRAM.

*A.2.2* **taclear.** This command will clear one, or both of the DRAM banks of memory to a specified color. The default is to clear Bank A to black. **taclear** was used mainly when about to load a new image, and when logging out from the workstation.

*A.2.3* **taread.** **taread** will write from the TAAC-1 DRAM to standard output in the form of a 32-bit image file (iff). The starting address and image size may be specified; the defaults are 2D address 0,0 and a size of 512 X 512 pixels. This command is most useful when saving data either for redisplay at a later time, or for conversion to another file format for printing purposes (covered in Appendix D). (As these files may become quite large, it is a good idea to compress the iff file to save disk space within the file server.)

*A.2.4* **taload.** **taload** performs an inverse function to that of **taread**: it will write an iff file to a specified location of the TAAC-1 DRAM, defaulting to 2D address 0,0 if no address is provided.

*A.2.5* **gettaac.** Available on the Louvre file server, **gettaac** provides an alternative to using **tainit** and **taload** (which display an iff file on the TAAC-1 video) by allowing display of a Utah rle file. Running **gettaac** will initialize the TAAC-1, along with storing/displaying a rle file to the TAAC-1; Appendix B gives an example of when the use of **gettaac** may be appropriate. The executable code for **gettaac** is in the **/usr/local/bin** directory of Louvre.

*A.2.6* **tadeb.** An extremely useful tool in this effort was the TAAC-1 debugger, **tadeb**. Through the use of this utility, one can load and run a TAAC-1 program, look at

the contents of memory, set breakpoints or single step, and to read and write memory. While **tadeb** was used a few times for single-stepping through a program, the main benefit gained from this tool was the access to memory contents. Figure 25 was very useful in this regard – it allowed for an easier transition from the 2D addressing used in the TAAC-1 programs to the 1D scheme required for memory access via **tadeb.** Another highlight was the multiple radix selection for viewing data stored in the TAAC-1 DRAM. It is strongly suggested that this section of the TAAC-1 User Guide be read by anyone attempting TAAC-1 programming.

### *A.3 Programming*

As the TAAC-1 is limited in the types of tasks it can perform, a decision to be made when developing programs for the application accelerator is whether to write a stand-alone TAAC-1 program (an example was the **colorblend.tc** program mentioned in Section A.1.2), or to integrate the TAAC-1 program with a program written for the host device. A host routine must be used if any of the following tasks are desired:

- Disk I/O,
- User interface,
- Window management

*A.3.1 Host/TAAC-1 Program Integration.* A handshaking protocol is most often used when integrating a host program with a TAAC-1 program. It is a simple process, and one which allows synchronization of the normally asynchronous host and TAAC-1 programs. As an example, a program was written for this effort which allowed results from the 2DFFT TAAC-1 process to be stored in a separate file - a task which required both a host program (file manipulation) and a TAAC-1 program (2DFFT process).

The host program written for this task is **store_data.c**, while the TAAC-1 program is **datastorage.tc**, both of which are listed in Appendix C. Figure 31 provides an overview of the handshaking protocol used between the two programs, which uses the setting and clearing of a flag to accomplish this task.

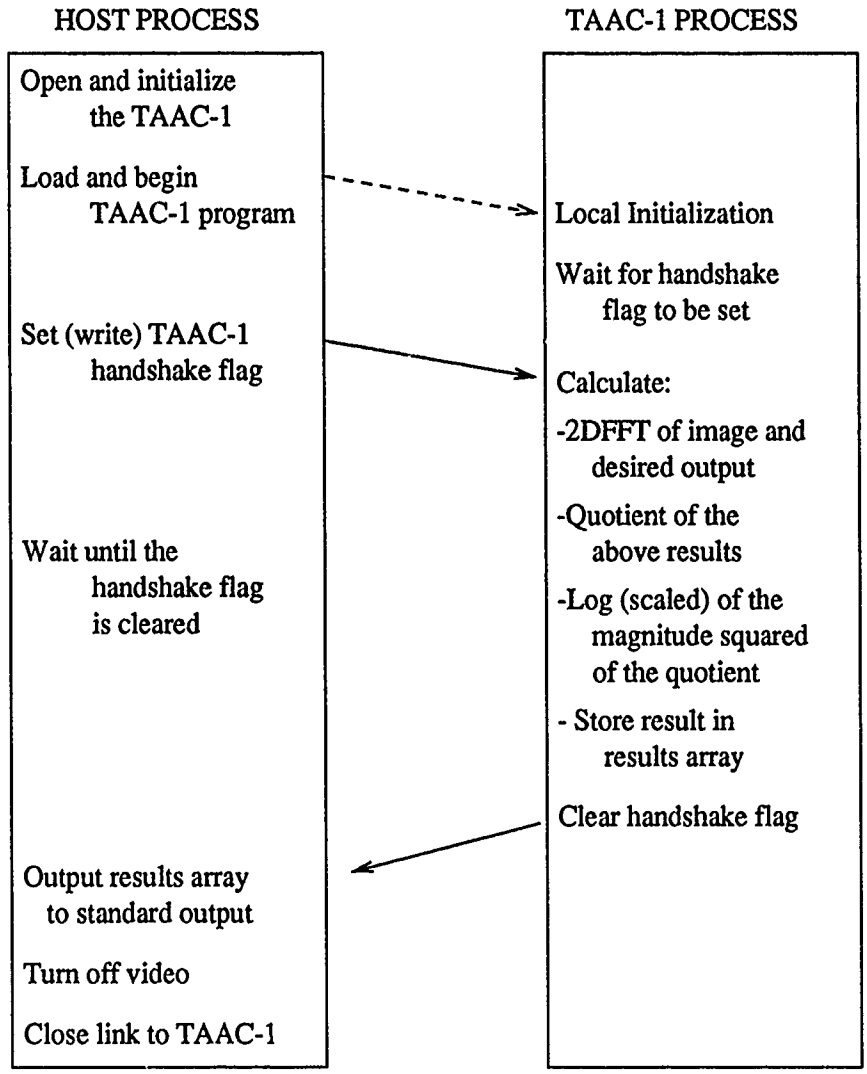



Figure 31. Handshaking Protocol Between Host and TAAC-1 (47:3-5)

Two other files, **store_data.h** and **Makefile (store_data/datastorage)**, were used in conjunction with **store_data.c** and **datastorage.tc. store_data.h** contains the **defines** used in the two programs, while **Makefile (store_data/datastorage)** directs the compilation and linkage specifically for **store_data.c** and **datastorage.tc.** The executable program created as a result of using this Makefile is **store_data.** The programs are well-commented and should provide enough insight to program integration to be used as basic examples of the handshaking protocol.

*A.3.2 TAAC-1 Stand-Alone Programs.* An alternative to integrating host programs with TAAC-1 routines is to run TAAC-1 programs as stand-alone units; of course, the desired programming tasks will dictate whether or not this method should be used. Many of these type of programs were used during the course of this thesis effort; some examples listed in Appendix C are **SquareIdealLPF.tc** and **CircIdealLPF.tc. Makefile (standalone)** is also included in Appendix C - however a note about its use should be added here.

As so many stand-alone programs were to used/tested during this project, a boilerplate Makefile was written under the assumption that all programs to be run would first be copied into the boilerplate file **a1.tc.** Then, using the command **make new_run, a1.tc** would be compiled and linked, with the executable code in the file **a1.abs.** The command **tarun a1.abs** was then used to run the program. This process was made even easier by creating aliases for the commands used quite often.

## *A.4 Libraries*

Several library types are available for use with TAAC-1 programs; the ones used for this effort are described here, along with a short listing of the library routines. Greater detail of these libraries is provided in the TAAC-1 Software Reference Manual (46).

*A.4.1 Graphics Library.* The major uses of the TAAC-1 graphics routines are to transform, shade, clip and render 2D and 3D images; most are written in assembly language for speed, with the expectation that the user will write a higher-level shell around the routine.

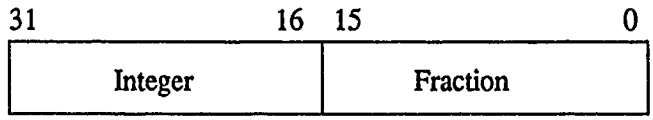



Figure 32. Integer/Fraction Form for Device Coordinate Space (47:11-40)

Programs using a graphics library routine must contain the include file **t_graphics.h** which has function prototypes for the graphics library functions along with structure and constant definitions for these functions.

*A.4.1.1 t_line2d/t_dcline2d.* These routines allow for drawing of 2D vectors - the difference between the two is that **t_dcline2d** interpolates between a beginning and ending shade, while **t_line2d** is a single shade. Starting points, ending points and the desired shade(s) should be specified. An important point to note is the number format for the vertex coordinates: they must be in device coordinate space and expressed in integer/fraction form as shown in Figure 32.

Numbers in the 2D addressing scheme can be converted to this form by simply multiplying by 65536 ($2^{16}$).

*A.4.1.2 t_erase/t_rect.* **t_erase** allows for quick erasure of a rectangular area of TAAC-1 memory to a specified 32-bit shade value. A restriction on **t_erase** is its limitation on block sizes being multiples of 256. Although slower than **t_erase**, **t_rect** will allow erasure of an arbitrarily-sized area.

*A.4.2 Image Processing Library.* These routines perform such image processing tasks as statistical analysis, algebraic and Boolean operations, and FFT processing. Any program using a image processing library routine must contain the include file **t_ipl.h** which, like **t_graphics.h**, contains function prototypes and structure/constant definitions.

As general note for these routines, two arguments used in many of the function calls

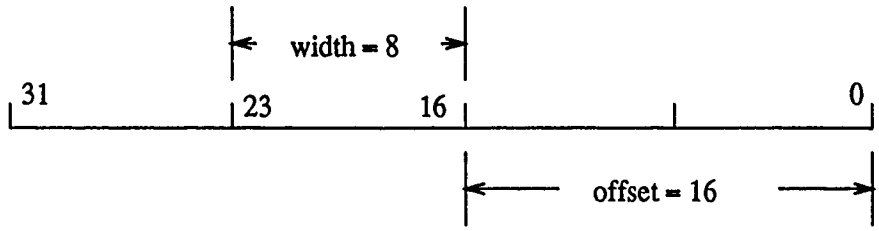



Figure 33. Example of Width and Offset Designation

are the **width** and **offset** arguments. These may be used to select a specific subset of the 32-bit word; as an example, if just the blue color bits (bits 16-23) are to be selected, an offset of 16 and a width of 8 would be chosen, as shown in Figure 33.

*A.4.2.1 t_copy.* **t_copy** copies a rectangular array of the TAAC-1 DRAM from one specified location to another. As many of the image processing routines write over the source data, this function is useful when source data must be saved besides along with being manipulated. Arguments include source and destination locations, and image size.

*A.4.2.2 t_divide/t_udivide/t_fdivide.* These routines will perform a pixel-by-pixel division of one image by another. **t_divide** is for signed integer division, **t_udivide** for unsigned integer division, and **t_fdivide** is used for single floating point division. Source locations for both original images must be specified as arguments, along with the destination location, width and offset for both sources and destination, and the number of rows and columns in the quotient. Similar routines exist for multiplication of images (**t_multiply/t_umultiply/t_fmultiply**).

*A.4.2.3 t_extrema/t_uextrema.* These routines return both the maximum and minimum pixel values of a source image, for either signed or unsigned images, respectively. This function comes in handy when determining the range of values that an image encompasses. Arguments include the source image location, the size of the image, width, and offset, along with pointers to the maximum and minimum pixel values found.

*A.4.2.4 t_fftinit.* Before using any of the TAAC-1 FFT routines, the TAAC-1 lookup table must first be initialized with FFT weights and bit-reversal indices. Therefore, the command **t_fftinit()** must be used prior to executing the first FFT domain function.

*A.4.2.5 t_fft2d/t_ifft2d.* The 2D forward discrete Fourier Transform on real integer source data is performed using the **t_fft2d** routine, which provides a complex, floating-point result. **t_ifft2d** will provide the inverse discrete Fourier Transform of that result, the response being in real-integer form. Some restrictions on both routines are that the image dimensions must be a power of two (identical for both source and destination), and that the width and offset specifications apply only to the destination data.

The algorithm used to perform the 2DFFT is a decimation-in-time, performed initially on each row of the data array, then done on each column of the row result. It is important to fully understand the storage method of the floating-point output – chosen to allow the output to occupy the same amount of memory space as the input. As discussed in Section A.1.2, the video display of the 2DFFT output is meaningless. It is therefore necessary to know where each component is located should evaluation of data be required.

Based on the relationship that,

$$\Re e[fft(i)] \equiv \Re e[fft(N-i)]$$

and,

$$\Im m[fft(i)] \equiv -\Im m[fft(N-i)],$$

symmetry can be evoked and the resultant array stored in an area of similar size to the input real array, even though nearly twice as many data elements are now available due to the inclusion of imaginary components.

Figures 34 and 35 demonstrate the difference between what might be labeled "normal" fft storage (Figure 34) and the type of fft storage used by the **t_fft2d** routine (Figure 35). In

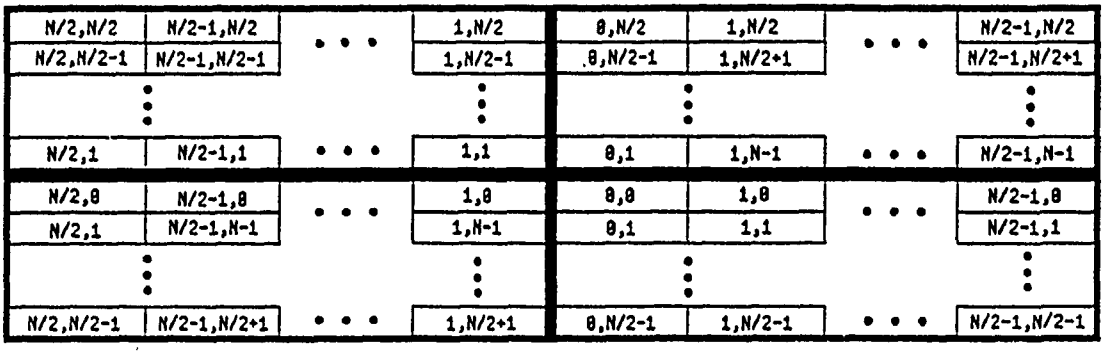



Figure 34. Example of Typical Storage of FFT Results

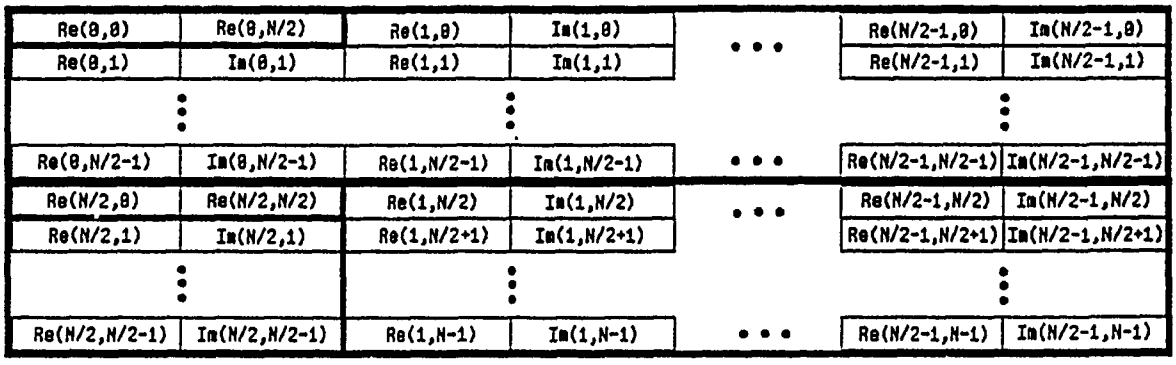



Figure 35. FFT Storage Scheme for the TAAC-1 **t_fft2d** Routine

the normal storage mode, the DC term is centered in the array, surrounded by components of the harmonics as they increase out to the Nyquist value ($\frac{N}{2}$).

Using the symmetry cited above, the right half of Figure 34 is used exclusively in Figure 35, with the upper right quadrant of Figure 34 being stored below the lower right quadrant in Figure 35. The following exceptions should be noted:

- Those terms without imaginary components (the DC, $(0, \frac{N}{2})$, $(\frac{N}{2},0)$, and $(\frac{N}{2}, \frac{N}{2})$ terms) have been grouped in twos at the beginning of the top and bottom half of the array.

- The column which runs from $(0, \frac{N}{2} - 1)$ to $(0, 1)$ is duplicated in the upper right and lower right quadrants of Figure 34; as such, it is replaced with the column $(\frac{N}{2}, 1) \ldots (\frac{N}{2}, \frac{N}{2} - 1)$ in Figure 35.

*A.4.2.6 t_log_spectrum.* One method to gain a better understanding of the 2DFFT output, is to reorganize the result into the form shown in Figure 34, and to scale the data in some way as to emphasize/deemphasize the differences between datum. **t_log_spectrum** accomplishes such organization by computing the $\log_{10}$ of the magnitude squared of the result, linearly scaled between 0 and the maximum value.

A discrepancy was noted when using **t_log_spectrum**: two passes are used to perform this routine, the first computes the square of the magnitudes and stores them in the right half of the destination area. The second pass computes the log of these values and performs the linear transform; it is within this second pass that the error was noted. As the final results are displayed, the right half of the destination displays the final result *and* the result from the first pass simultaneously, making it extremely difficult to view the display of that data.

As symmetry is used between the right and left halves of the result of **t_log_spectrum**, viewing the left half alone might suffice for some applications. However, in order to view the entire result as was intended, a separate program was written which performs as **t_log_spectrum** was initially meant to. The programs which perform the **t_log_spectrum** function is the **store_data.c/datastorage.tc** combination previously discussed in Section A.3.1.

*A.4.2.7 t_multconst.* Where Section A.4.2.2 dealt with manipulation of two images, **t_multconst** allows for multiplication of each pixel in an image by a signed constant.

*A.4.2.8 t_threshold/t_uthreshold.* This routine is used to segment an image, based on gray-scale values. Each pixel of the image is compared to a designated threshold value and, based on the user-selected flag, the pixel may be manipulated in a number

of ways. **t_threshold** is used for signed compares and thresholds, and **t_uthreshold** for unsigned operators.

*A.4.3 Mathematics Library.* When mathematical operations are required on entire image arrays, the routines in this library are most useful. Again, should any of these routines be used within a program, the include file **t_math.h** must be used.

*A.4.3.1 t_exp/t_log10/t_pow.* Some of the more useful, most commonly used routines deal with exponentials (**t_exp**), logarithms (**t_log10**), and raising one argument to the power of a second argument (**t_pow**). There are floating-point and double precision forms of each, while there is also a routine which returns the natural logarithm of an argument (bf t_log).

*A.4.3.2 t_sqrt.* Double precision (**t_dsqrt**) or floating-point (**t_sqrt**) square roots of arguments may be obtained by using these routines.

## Appendix B. *Display of Anomaly Images*

The steps used to create and display anomaly images are as follows:

1. Use **touchup** to create a rasterfile of the desired image (see the man pages available on the Sun-4 for a detailed description on the usage of **touchup** and all commands provided in this Appendix).
2. Change **touchup**'s rasterfile output (file1.ras) to a rle file via: **rastorle file1.ras > file2.rle.**
3. If file2.rle were to be displayed now, it would be an upside-down version of the original **touchup** output. Convert image back to original coordinates via: **rleflip -v file2.rle > file3.rle**
4. Display the rle file on the TAAC video terminal: **gettaac file3.rle**

# Appendix C. *Software Compendium*

This appendix will list those programs written specifically for this thesis effort. Though they have not been written with the mainstays of software engineering in mind, they are fairly well commented and should be easy to use and/or modify.

## *C.1 Host/TAAC-1 Program Integration Examples*

These programs were referenced from Section A.3.1 of Appendix A. They allow for reading of data from the TAAC-1 DRAM into an array, the contents of which may then be read into a file.

### *C.1.1 store_data.c*

```
/*****************************************************************/
/* FILE : store_data.c, located in the ~robern/host_and_taac     */
/*      directory of Louvre                                      */
/* DATE : 31 August 1990                                         */
/* VERSION : 1.0                                                 */
/* AUTHOR : C. Oberndorf                                         */
/* DESCRIPTION : This program is used in conjunction with        */
/*      datastorage.tc.  Its main purpose is to save data stored*/
/*      in the TAAC-1 DRAM.  The data is the floating-point      */
/*      result of the 2DFFT of a particular image after it has   */
/*      gone through a process of taking the log of the          */
/*      magnitude squared (via datastorage.tc).                  */
/* NOTE :  The float variable results[][] is limited in size.    */
/*      A 128 X 128 array was too large for the TAAC-1, so the   */
/*      array was halved and run twice to obtain the entire      */
/*      array of data.                                           */
/* FUNCTIONS :                                                   */
/*      taacstartup() - opens, initializes, loads and runs the   */
/*              TAAC-1; taken directly from a sample program in */
/*              the TAAC-1 User Guide                            */
/*      write_error() - informs user of any difficulties in      */
/*              writing to the TAAC-1; taken directly from a     */
```

```
/*              sample program in the TAAC-1 User Guide          */
/*      read_error() - informs user of any difficulties in       */
/*              reading from the TAAC-1; taken directly from a   */
/*              sample program in the TAAC-1 User Guide          */
/*      ta_zero(tahndl) - host routine which stops the TAAC-1    */
/*              processor clock and zeroes the program counter   */
/*              and stack pointer                                */
/*      ta_set_display (tahndl, TA_DEFAULT) - sets the TAAC-1    */
/*              display mode; the use of TA_DEFAULT in           */
/*              conjunction with /taac/hardware/taconfig.(host)  */
/*              selects the hardware configuration regarding     */
/*              single vs. dual monitors and TAAC-1 vs. external*/
/*              sync.                                            */
/*      ta_close(tahndl) - closes the virtual memory link        */
/*              between the host and the TAAC-1                  */
/* HISTORY:  This code was originally designed as part of a      */
/*      thesis effort, Analysis of Visual Illusion Using Gabor   */
/*      Filters.  The 'guts' of this program is taken from a     */
/*      sample TAAC-1 program, colors.c, found in the TAAC-1     */
/*      User Guide, Chapter 3.                                   */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,      */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;         */
/*      TAAC-1 Application Accelerator: Software Reference       */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/*****************************************************************/
#include <stdio.h>
#include <taac1/taio.h>          /* TAAC-1 host include file */

#include "store_data.h"          /* store_data defines */
#include "datastorage_map.h"     /* global defines from tamakedef */

main ()
{

    TA_HANDLE *tahndl;           /* Handle returned from ta_open */
    TA_HANDLE *taacstartup ();   /* Routine to open, init TAAC-1 */
    void write_error ();         /* Informs if write error occurs*/
    void read_error ();          /* Informs if read error occurs */
    int tastatus;
    int ioflag,a,b;
    float results[128][64];      /* Holds data created by TAAC-1 */
```

```
/*
Open, initialize, load, run TAAC-1
*/
    tahndl = taacstartup ();

/*
Set handshaking flag, and write to TAAC-1
*/
    ioflag = 1;
    if ((tastatus = ta_write (tahndl, &ioflag, sizeof(ioflag),
           TC_ioflag)) != sizeof(ioflag)) {
                write_error (tastatus);
            }

/*
Wait for TAAC-1 to finish, signified by ioflag reset to zero
*/
    while (ioflag) {
        if ((tastatus = ta_read (tahndl, &ioflag, sizeof(ioflag),
                TC_ioflag)) != sizeof(ioflag)) {
            read_error (tastatus);
        }
    }

/*
Get results[][] from TAAC-1
*/
    if ((tastatus = ta_read (tahndl, results, sizeof(results),
                TC_results)) != sizeof(results))
             read_error (tastatus);

/*
Direct results[][] to standard output
*/
    for (a=32;a<64;a++) {
        for(b=32;b<96;b++)
           printf("%f ",results[b][a]);
        printf("\n");
    }

/*
Cause TAAC-1 processor to halt and loop at PC = 0
```

```
*/
    ta_zero(tahndl);

/*
Then turn off TAAC-1 video
*/
    ta_set_display (tahndl, TA_DEFAULT);

/*
And close TAAC-1 link
*/
    ta_close(tahndl);

} /* end main */

/******** taacstartup ********/

/*
Open TAAC-1 and begin program execution
*/

TA_HANDLE *taacstartup ()
{
    TA_HANDLE *tahndl;        /* handle returned from ta_open() */
    int tastatus;             /* returned status */

/*
Open TAAC-1 link
*/
    if ((tahndl = ta_open (0)) == NULL) {
        fprintf (stderr, "Error opening TAAC-1\n");
        exit (-1);
    }

/*
Initialize the TAAC-1 lookup table
*/
    if ((tastatus = ta_init (tahndl)) != TA_SUCCESS) {
        fprintf (stderr, "ta_init failed, returned %d\n", tastatus);
        exit (-1);
    }
```

```
/*
Set up the TAAC-1 display mode
*/
    if ((tastatus = ta_set_display (tahndl, TA_ON)) != TA_SUCCESS) {
        fprintf (stderr, "ta_set_display failed, returned %d\n", tastatus);
        exit (-1);
    }
/*
The TAAC-1 executable code is loaded, the PC and SP reset,
and code execution begun
*/
    if ((tastatus = ta_runm (tahndl)) != TA_SUCCESS) {
        fprintf (stderr, "ta_runm failed, returned %d\n", tastatus);
        exit (-1);
    }
    return (tahndl);
}



/******** write_error ********/

void write_error(status)
int status;
{
    fprintf(stderr,"ta_write failed, returned  %d\n",status);
    exit(-1);
}


/******** read_error ********/

void read_error(status)
int status;
{
    fprintf(stderr,"ta_read failed, returned  %d\n",status);
    exit(-1);
}
```

### *C.1.2 datastorage.tc*

```
/*****************************************************************/
/* FILE : datastorage.tc, located in the ~robern/host_and_taac   */
/*      directory of Louvre                                      */
/* DATE : 31 August 1990                                         */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  This program is used in conjunction with       */
/*      store_data.c.  Its purpose is to perform a 2DFFT on      */
/*      an image, with the result a floating-point number.  A    */
/*      knowledge of the layout of the 2DFFT result is needed    */
/*      for an understanding of the data manipulation taking     */
/*      place in this program.  The data then goes through a     */
/*      process of taking the log of the magnitude squared.      */
/*      datastorage.tc is used in conjunction with store_data.c, */
/*      which saves results[][] to a separate file.              */
/* NOTE :  The float variable results[][] is limited in size.    */
/*      A 128 X 128 array was too large for the TAAC-1, so the   */
/*      array was halved and run twice to obtain the entire      */
/*      array of data. To obtain the top half of the 128 X 128 */
/*      results array, datastorage.tc should be loaded with      */
/*      tophalf.tc which manipulates the correct portion of the */
/*      2DFFT output.  Similarly, load bottomhalf.tc into        */
/*      datastorage.tc to obtain the lower half of the array.    */
/*      Two separate files were used so that only one Makefile   */
/*      was necessary (keeps things in terms of store_data.c     */
/*      and datastorage.tc only).                                */
/* FUNCTIONS :                                                   */
/*      CMPLX_UPDATE(src, dest1, dest2) - reads data from source*/
/*              (src) location and calculates the log of the     */
/*              magnitude squared;  keeps track of the minimum   */
/*              value obtained for later scaling of data; due to*/
/*              symmetry, data is saved to results[][] in two    */
/*              separate locations                               */
/*      set_ac(AC_LDALL, src) - sets the Address Count Register */
/*              (AC) to the source (src) address                 */
/*      read_ac() - reads data from the address in the AC        */
/*      write_ac(response) - writes value of response to the     */
/*              address stored in the AC                         */
/*      t_log10(response) - calculates base 10 logarithm         */
/*      t_fftinit() - initializes the TAAC-1 look-up table with */
/*              FFT weights                                      */
```

```
/*      t_fft2d() - performs 2DFFT on the SRC integer data,     */
/*              placing the floating-point result at the DEST   */
/*      t_fdivide() - pixel by pixel division of SRC1 by SRC2   */
/*              placing the floating-point result at the DEST   */
/* HISTORY:  This code was originally designed as part of a     */
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.                                                */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,     */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;        */
/*      TAAC-1 Application Accelerator: Software Reference      */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/***************************************************************/
/*  add libraries necessary to run various subroutines */
#include </taac/include/taac1/t_ipl.h>
#include </taac/include/taac1/t_math.h>
#include </taac/include/taac1/builtin.h>
#include </taac/include/taac1/t_graphics.h>

#include "store_data.h" /*  store_data defines  */

/*
The following subroutine calculates the magnitude squared
and the logarithm (base 10) of that result.  The result is
stored in results[][].  The minimum value is maintained for
later scaling use.
*/
#define CMPLX_UPDATE(src,dest1,dest2) \
    { \
    set_ac(AC_LDALL,src); \
    real = asfloat(read_ac()); \
    upd_ac(INC_COL); \
    imag = asfloat(read_ac()); \
    response=real*real+imag*imag; \
    results[y1][z1]=t_log10(response); \
    results[y2][z2]=t_log10(response); \
    if(results[y1][z1]<min) \
        min=results[y1][z1]; \
    else if(results[y2][z2]<min) \
        min=results[y2][z2]; \
    set_ac(AC_LDALL,dest1); \
    write_ac(aslong(response)); \
    set_ac(AC_LDALL,dest2); \
```

```
    write_ac(aslong(response)); \
    }

/* Globals */
    int ioflag=0;
    float results[128][64];

main()
{
register RD int         sstart, dstart1, dstart2,
                        sp1, dp1, dp2,;

register float          min=0., response, real, imag;

register int            rows, cols;

int                     SRC, DEST, nyquist, y1, y2, z1, z2,
                        OldAM;

/*
Loop until host sets ioflag=1
*/
    while(ioflag==0);
/*
Initialize TAAC-1 look-up table with FFT weights, do 2DFFT
of original image, then desired output, finishing by
dividing the FFT results to obtain the FFT of the transfer
function.
*/
    t_fftinit();
    t_fft2d(0,0,SAVEFFT1_X,SAVEFFT1_Y,SIZE,SIZE,WIDTH,OFFSET);
    t_fft2d(0,IMAGE,SAVEFFT2_X,SAVEFFT2_Y,SIZE,SIZE,WIDTH,OFFSET);
    t_fdivide(SAVEFFT2_X,SAVEFFT2_Y,SAVEFFT1_X,SAVEFFT1_Y,
         QUOTIENT_X,QUOTIENT_Y,SIZE,SIZE);

/*
Save AM register mode
*/
    OldAM = input(RD_AM);

/*
Set AM register for 2d addressing mode
```

```
*/
    output(WR_AM,TA_AM2D);

/*
Create 2d addresses for source(s) and destination
*/
    SRC = ((QUOTIENT_Y << 16) | QUOTIENT_X);
    DEST = ((SAVEFILTER_Y << 16) | SAVEFILTER_X);

/*
Compute parameters to be used in data transfer
*/
    nyquist = SIZE/2;
    cols = rows = nyquist - 1;

/*
This section puts rows 1,1...(nyquist-1),1 through
1,(nyquist-1)...(nyquist-1),(nyquist-1) at either the upper
left quadrant (tophalf.tc) or the lower right (bottomhalf.tc
at upper left quadrant (dp2) and lower right (dp1)
*/
    sstart=SRC+2;
    dstart1=DEST+nyquist*YADDRINC+nyquist+1;
    dstart2=DEST+nyquist*YADDRINC+nyquist-1;
    z1=z2=nyquist;
    loop(rows) {
        sp1=sstart+=YADDRINC;
        dp1=dstart1+=YADDRINC;
        dp2=dstart2-=YADDRINC;
        y1=y2=nyquist-1;
        z1=z2-=1;
        loop(cols) {
            CMPLX_UPDATE(sp1,dp1,dp2);
            sp1+=2;
            dp1+=1;
            dp2-=1;
            y1=y2-=1;
        }
    }

/*
This section puts row 1,nyquist...(nyquist-1),nyquist at
```

```
both the upper left quadrant and the upper right (for
tophalf.tc) or skips it completely (for bottomhalf.tc)
*/
    sp1=sstart+=YADDRINC;
    dp2=dstart2-=YADDRINC;
    dp1=dp2+2;
    dstart1=dp1;
    z1=z2=0;
    y1=nyquist+1;
    y2=nyquist-1;
    loop(cols) {
        CMPLX_UPDATE(sp1,dp1,dp2);
        sp1+=2;
        dp1+=1;
        dp2-=1;
        y1+=1;
        y2-=1;
    }

/*
This part does rows 1,(nyquist+1)...15,(nyquist+1)
through 1,(size-1)...15,(size-1) for the upper right
quadrant (tophalf.tc) or the lower left (bottomhalf.tc)
*/
    dstart2=DEST+SIZE*YADDRINC+rows;
    z1=z2=0;
    loop(rows) {
        sp1=sstart+=YADDRINC;
        dp1=dstart1+=YADDRINC;
        dp2=dstart2-=YADDRINC;
        z1=z2+=1;
        y1=y2=nyquist+1;
        loop(cols) {
            CMPLX_UPDATE(sp1,dp1,dp2);
            sp1+=2;
            dp1+=1;
            dp2-=1;
            y1=y2+=1;
        }
    }
/*
The DC term is done first then the row of 0,1...0,(nyquist-1)
```

```
for the upper right quadrant or the lower right
*/
    sp1=SRC;
    dp1=dp2=DEST+nyquist*YADDRINC+nyquist;
    set_ac(AC_LDALL,SRC);
    real = asfloat(read_ac());
    response=real*real;
/*
Next line commented out for tophalf.tc
*/
/*  results[nyquist][64]=t_log10(response);*/
    set_ac(AC_LDALL,dp1);
    write_ac(aslong(response));
    y1=y2=z1=z2=nyquist;
    loop(rows) {
        sp1+=YADDRINC;
        dp1+=YADDRINC;
        dp2-=YADDRINC;
        z1=z2-=1;
        CMPLX_UPDATE(sp1,dp1,dp2);
    }
/*
Row of 1,0...(nyquist-1),0 done twice, once for the lower
left quadrant, once for the lower right; nothing done for
tophalf.tc
*/
    sp1=SRC;
    dp1=dp2=DEST+nyquist*YADDRINC+nyquist;
    y1=y2=z1=z2=nyquist;
    loop(rows) {
        sp1+=2;
        dp1+=1;
        dp2-=1;
        y1+=1;
        y2-=1;
/*
Next line commented out for tophalf.tc
*/
/*        CMPLX_UPDATE(sp1,dp1,dp2);*/

    }
```

```
/*
Start with term nyquist,0 (for bottomhalf.tc) then row
nyquist,1...nyquist,(nyquist-1) being sent to the
upper left and the bottom left quadrants
*/
    sp1=SRC+nyquist*YADDRINC;
    dp1=dp2=DEST+nyquist*YADDRINC;
    set_ac(AC_LDALL,sp1);
    real = asfloat(read_ac());
    response=real*real;
/*
Next line commented out for tophalf.tc
*/
/*  results[0][64]=t_log10(response);*/
    set_ac(AC_LDALL,dp1);
    write_ac(aslong(response));
    y1=y2=0;
    z1=z2=nyquist;
    loop(rows) {
        spi+=YADDRINC;
        dp1+=YADDRINC;
        dp2-=YADDRINC;
        z1=z2-=1;
        CMPI.X_UPDATE(sp1,dp1,dp2);
    }

/*
0,nyquist done for tophalf.tc
*/
    set_ac(AC_LDALL,SRC+1);
    real = asfloat(read_ac());
    response = real*real;
    results[nyquist][0]=t_log10(response);
    set_ac(AC_LDALL,DEST+nyquist);
    write_ac(aslong(response));
/*
nyquist,nyquist done for tophalf.tc
*/
    set_ac(AC_LDALL,SRC+nyquist*YADDRINC+1);
    real = asfloat(read_ac());
    response = real*real;
    results[0][0]=t_log10(response);
```

```
    set_ac(AC_LDALL,DEST);
    write_ac(aslong(response));

/*
scale results[][] terms upwards by the minimum value
*/
   for(y1=0;y1<128;y1++)
       for(z1=0;z1<64;z1++)
          results[y1][z1]=results[y1][z1]-min;

/*
reset addressing mode
*/
    output(WR_AM,OldAM);
/*
Tell host program that work is done
*/
    ioflag=0;
    return(IPL_SUCCESS);
}
```

### *C.1.3 store_data.h*

```
/*****************************************************************/
/* FILE : store_data.h, located in the ~robern/host_and_taac     */
/*      directory of Louvre                                      */
/* DATE : 31 August 1990                                         */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  This program is used in conjunction with       */
/*      datastorage.tc and store_data.c.  It holds the defines   */
/*      used by those programs.                                  */
/* FUNCTIONS : none                                             */
/* HISTORY:  This code was originally designed as part of a      */
/*      thesis effort, Analysis of Visual Illusion Using Gabor   */
/*      Filters.                                                 */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,      */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;         */
/*      TAAC-1 Application Accelerator: Software Reference       */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/*****************************************************************/

/*  pixel width and length  */
#define SIZE 128

/*  x,y location of first FFT result  */
#define SAVEFFT1_X 128
#define SAVEFFT1_Y 0

/*  y location of the second image  */
#define IMAGE 256

/*  x,y location of second FFT result  */
#define SAVEFFT2_X 128
#define SAVEFFT2_Y 256

/*  x,y location of the quotient  */
#define QUOTIENT_X 256
#define QUOTIENT_Y 256

/*  x,y location of the filter output  */
#define SAVEFILTER_X 384
#define SAVEFILTER_Y 384
```

```
/*  R,G,B selection  */
#define WIDTH 16
#define OFFSET 0
```

### *C.1.4 Makefile (store_data.c/datastorage.tc)*

```
/*************************************************************/
/* FILE : Makefile (store_data/datastorage), located in the  */
/*      ~robern/host_and_taac directory of Louvre            */
/* DATE : 31 August 1990                                     */
/* VERSION : 1.0                                             */
/* AUTHOR : R. Oberndorf                                     */
/* DESCRIPTION :  Directs compilation and linkage specifically */
/*      for store_data.c and datastorage.tc.  Executable program*/
/*      is 'store_data'.                                     */
/* FUNCTIONS :                                               */
/*      cc - C compiler for host program                     */
/*      taabs2o - converts a TAAC-1 executable in absolute file */
/*              (.abs) format to a host object file so that it  */
/*              can be linked into a host program            */
/*      tamakedef - generates an include file for a host program*/
/*              containing the addresses of all global symbols  */
/*              in a TAAC-1 program                          */
/* HISTORY:  This code was originally designed as part of a  */
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.  It was based on an example Makefile for the   */
/*      programs colors.c/colorbar.tc, page 3-25 of the TAAC-1  */
/*      User's Guide                                         */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,  */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;     */
/*      TAAC-1 Application Accelerator: Software Reference   */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/*************************************************************/
# makefile for store_data.c/datastorage.tc

# include the predefined rules for building TAAC-1 programs
include /usr/include/taac1/makerules

# define the constants
CFLAGS = -O                     # flags for cc
LIBS = -ltaac1                  # libraries for hos. program
OBJS = store_data.o datastorageabs.o     # object files for host program

TCFLAGS = -c -fsingle           # flags for tacc
TLFLAGS =                       # flags for talink
TLIBS = -ltaac1                 # libraries for TAAC-1 program
TOBJS = datastorage.obj         # object files for TAAC-1 program
```

```
# compile (implicit) and link for host program
store_data: $(OBJS)
        $(CC)  $(CFLAGS) -o store_data $(OBJS) $(LIBS)

# dependencies for host program
store_data.o: store_data.h datastorage_map.h

#create a .o file from a TAAC-1 .abs file (TAABS20 is defined in makerules)
datastorageabs.o: datastorage.abs
        $(TAABS20) datastorage.abs datastorageabs.o

#create a .h file from the TAAC-1 .map file (TAMAKEDEF is defined in makerules)
datastorage_map.h: datastorage.map
        $(TAMAKEDEF) -d -c datastorage.map datastorage_map.h

# compile (implicit) and link TAAC-1 program (TALINK is defined in makerules)
# makerules also describes how to make a .obj file from a .tc file
datastorage.abs datastorage.map: $(TOBJS)
         $(TALINK) $(TLFLAGS) $(TOBJS) -o datastorage.abs $(TLIBS)

# dependencies for TAAC-1 program
datastorage.obj: store_data.h
```

## C.2 Stand-Alone TAAC-1 Program Examples

Sections A.1.2 and A.3.2 of Appendix A made reference to programs written which did not interact with a host program. Those examples are provided here.

### C.2.1 colorblend.tc

```
/*****************************************************************/
/* FILE : colorblend.tc, located in the ~robern/standalone_taac */
/*      directory of Louvre                                      */
/* DATE : 31 August 1990                                         */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  A short program whose purpose is to provide a */
/*      quick example of the color representation of a range of */
/*      integers.  The START and FINISH defines can be changes  */
/*      to whatever interval desired.                            */
/* FUNCTIONS :                                                   */
/*      t_dcline2d() - draws a shade-interpolated line from      */
/*            between two points with the shade varying from     */
/*            START to FINISH                                    */
/* HISTORY:  This code was originally designed as part of a      */
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.                                                 */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,      */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;         */
/*      TAAC-1 Application Accelerator: Software Reference       */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/*****************************************************************/
#include </taac/include/taac1/t_graphics.h>

/*
START = black and FINISH = white;  these may be changed as
desired keeping in mind that bits 0-7 are the red channel,
bits 8-15 the green channel, and 16-24 the green channel
*/
#define START 0x00000000
#define FINISH 0x00ffffff

main()
{
```

```
    int  i;
/*
This example will display a 128 X 128 pixel color blend
beginning at 2D address 0,0 with the START color and
ending at 2D address 127,127 with the FINISH color.
*/
for(i=0;i<128;i++)
    t_dcline2d(i*65536.,0.*65536,i*65536.,127.*65536.,START,FINISH);
}
```

### *C.2.2 SquareIdealLPF.tc*

```
/*****************************************************************/
/* FILE : SquareIdealLPF.tc, located in the                      */
/*      ~robern/standalone_taac directory of Louvre              */
/* DATE : 31 August 1990                                         */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  This program performs the 2DFFT of an image,   */
/*      filters the result with a square low-pass filter, and    */
/*      does an inverse 2DFFT on the filtered data.  The guts of*/
/*      the filtering process is based on the lowpass.tc program*/
/*      located in the /taac/src/taaclib/image directory of the */
/*      Sun-4 Workstations.  The filters available in the TAAC-1*/
/*      library are Butterworth type filters;  it was desired to*/
/*      have a more ideal filter, hence this program.            */
/* NOTE :  The section of Appendix A which deals with t_fft2d    */
/*      contains a figure which provides the data layout of the */
/*      t_fft2d result.  That figure, used in conjunction with   */
/*      the comments in this program, should provide sufficient */
/*      detail in understanding the "why's and wherefore's" of   */
/*      this program.                                            */
/* FUNCTIONS :                                                   */
/*      CMPLX_UPDATE(src, dest) - performs the actual filtering */
/*              process on the DRAM data.  Takes the data stored*/
/*              at the source (src) address, multiplies it by    */
/*              the filter response, and stores the result at    */
/*              the destination (dest) address                   */
/*      set_ac(AC_LDALL, src) - sets the Address Count Register */
/*              (AC) to the source (src) address                 */
/*      read_ac() - reads data from the address in the AC        */
/*      write_ac(response) - writes value of response to the     */
/*              address stored in the AC                         */
/*      t_fftinit() - initializes the TAAC-1 look-up table with */
/*              FFT weights                                      */
/*      t_fft2d() - performs 2DFFT on the SRC integer data,      */
/*              placing the floating-point result at the DEST    */
/*      t_copy() - copies data from one area of the DRAM to      */
/*              another.  Saves original data.                   */
/*      t_ifft2d() - performs the inverse 2DFFT process on the   */
/*              SRC floating-point data, and stores the integer */
/*              result at DEST.                                  */
/* HISTORY:  This code was originally designed as part of a      */
```

```
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.                                                 */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,     */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;        */
/*      TAAC-1 Application Accelerator: Software Reference      */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/*************************************************************/
/*  add libraries necessary to run various subroutines */
#include </taac/include/taac1/t_ipl.h>
#include </taac/include/taac1/builtin.h>
#include </taac/include/taac1/t_graphics.h>

/*  image size in pixels */
#define SIZE 512
/*  multiplied by GAIN to get the filter extremes  */
#define LOW_RESPONSE 0
#define HIGH_RESPONSE 1
/*  acts as contrast variance  */
#define GAIN 0.04
/*  upper bandwidth factor: cutoff harmonic = BW * SIZE / 2  */
#define BW 0.0625
/*  x,y location of fft results   */
#define SAVEFFT_X 512
#define SAVEFFT_Y 0
/*  x,y location of low pass filter results  */
#define SAVEFILTER_X 512
#define SAVEFILTER_Y 512
/*  x,y location of inverse-fft results   */
#define RESULT_X 0
#define RESULT_Y 512
/*  RGB determination  */
#define WIDTH 16
#define OFFSET 0

/*
The following subroutine multiplies the filter response by the
fft values in the source location and stores the result in
the destination addresses.
*/
#define CMPLX_UPDATE(src,dest) \
    { \
    set_ac(AC_LDALL,src); \
```

```
    real = response*asfloat(read_ac()); \
    upd_ac(INC_COL); \
    imag = response*asfloat(read_ac()); \
    set_ac(AC_LDALL,dest); \
    write_ac(aslong(real)); \
    upd_ac(INC_COL); \
    write_ac(aslong(imag)); \
    }

main()
{
    register RD int     sulq1, sulq2, sllq1, sllq2,
                        dulq1, dulq2, dllq1, dllq2,
                        sp1, sp2, sp3, sp4,
                        dp1, dp2, dp3, dp4;

    register RC int     DiagInc1, DiagInc2,
                        ysq;

    register float      dist, A1, A2, bandwidth,
                        response, real, imag;

    register int        i, r, c,
                        rows, cols;
    int                 SRC, DEST,nyquist,
                        OldAM;

/*
Initialize the TAAC-1 look-up table with FFT weights, and perform
the 2DFFT of the original image
*/
    t_fftinit();
    t_fft2d(0,0,SAVEFFT_X,SAVEFFT_Y,SIZE,SIZE,WIDTH,OFFSET);

/*
Save AM register mode
*/
    OldAM = input(RD_AM);

/*
Set AM register for 2d addressing mode
*/
```

```
    output(WR_AM,TA_AM2D);

/*
Create 2d addresses for source(s) and destination
*/
    SRC = ((SAVEFFT_Y << 16) | SAVEFFT_X);
    DEST = ((SAVEFILTER_Y << 16) | SAVEFILTER_X);

/*
Compute filter transfer function parameters
*/
    nyquist = SIZE/2;
    bandwidth = BW*nyquist;

/*
The filtering method which follows is based on the figure shown
in Appendix A (in the section dealing with t_fft2d) which gives
the layout of the t_fft2d result.  Much of the method used here
is based on the symmetry between the upper and lower halves of
the t_fft2d result.
*/
/*
First process elements along diagonal lines of symmetry going
from the top left to the bottom right in the top half of the
result, and from the bottom left to top right for the bottom
half.
Initialize variables and pointers: the DiagIncs will increment
the pointers diagonally
*/
    DiagInc1 = YADDRINC + 2;
    DiagInc2 = YADDRINC - 2;
/*
Starting points for the upper half source (sulq)
and destination (dulq):
*/
    sulq1 = SRC;
    dulq1 = DEST;
/*
Starting points for the lower half source (sllq)
and destination (dllq):
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC;
```

```
    dllq1 = DEST + (SIZE-1)*YADDRINC;

    rows = nyquist - 1;
    r = 0;
    loop(rows)
        {
/*
Increment upper left quadrant pointers (moving diagonally top
left to bottom right)
*/
        sulq1 += DiagInc1;
        dulq1 += DiagInc1;
/*
Decrement lower left quadrant pointers (moving diagonally bottom
left to top right
*/
        sllq1 -= DiagInc2;
        dllq1 -= DiagInc2;
        r++;
/*
The filtering determination is based on whether the harmonic
is above or below the cutoff frequency (bandwidth)
*/
        if (r <= bandwidth)
                response = HIGH_RESPONSE*GAIN;
        else
                response = LOW_RESPONSE*GAIN;
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sllq1,dllq1);
        }

/*
Now process elements that are symmetric about diagonals
just done.  At the end of this section, the entire t_fft2d
result will have been filtered EXCEPT the top row of components,
the bottom row, and the first two columns on the left.
Initialize variables and pointers:
*/
    r = 0;
    rows = nyquist - 2;
    cols = nyquist - 1;
/*
```

```
Two pointers are used for each half - each of the two works on
one a separate side of the diagonal.  First the top half pointers
are initialized, sulq1 and dulq1 working on the right side of the
diagonal, sulq2 and dulq2 on the left side:
*/
    sulq1 = SRC + 2;
    dulq1 = DEST + 2;
    sulq2 = SRC + YADDRINC;
    dulq2 = DEST + YADDRINC;
/*
Then the bottom half pointers are initialized, sllq1 and dllq1
working on the right side of the bottom diagonal, sllq2 and dllq2
on the left side
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC + 2
    dllq1 = DEST + (SIZE-1)*YADDRINC + 2;
    sllq2 = SRC + (SIZE-2)*YADDRINC;
    dllq2 = DEST + (SIZE-2)*YADDRINC;

    loop(rows)
        {
        r++;
        ysq = r*r;
        cols--;
        c = r;
*
Update upper half pointers downward along diagonal
*/
        sp1 = sulq1 += DiagInc1;
        dp1 = dulq1 += DiagInc1;
        sp2 = sulq2 += DiagInc1;
        dp2 = dulq2 += DiagInc1;
/*
Update lower half pointers upward along diagonal
*/
        sp3 = sllq1 -= DiagInc2;
        dp3 = dllq1 -= DiagInc2;
        sp4 = sllq2 -= DiagInc2;
        dp4 = dllq2 -= DiagInc2;

        loop (cols)
            {
```

```
            c++;
/*
Again, the filtering determination is based on the harmonics
in both directions being compared to the cutoff frequency
*/
            if ((r <= bandwidth) && (c <= bandwidth))
                response = HIGH_RESPONSE*GAIN;
            else
          response = LOW_RESPONSE*GAIN;
            CMPLX_UPDATE(sp1,dp1);
            CMPLX_UPDATE(sp2,dp2);
            CMPLX_UPDATE(sp3,dp3);
            CMPLX_UPDATE(sp4,dp4);
/*
Increment upper half, right side pointer by columns
*/
            sp1 += 2; dp1 += 2;
/*
Increment upper half, left side pointer by rows
*/
            sp2 += YADDRINC; dp2 += YADDRINC;
/*
Increment lower half, right side pointer by columns
*/
            sp3 += 2; dp3 += 2;
/*
Decrement lower half, left side pointer by rows
*/
            sp4 -= YADDRINC; dp4 -= YADDRINC;
            }
        }
/*
Next, the top row of the result, the bottom row, and the upper
half left two columns will be filtered.
Initialize pointers and variables: sulq1 works on the top row,
sulq2 on the left two columns, and sllq1 on the bottom row.
*/
    sulq1 = sulq2 = SRC;
    dulq1 = dulq2 = DEST;
    sllq1 = SRC + (SIZE-1)*YADDRINC;
    dllq1 = DEST + (SIZE-1)*YADDRINC;
    c = 0;
```

```
    loop (nyquist - 1)
        {
/*
Move sulq1 and sllq1 to the right, and sulq2 downward
*/
        sulq1 += 2; /* by columns */
        dulq1 += 2; /* by columns */
        sulq2 += YADDRINC; /* by rows */
        dulq2 += YADDRINC; /* by rows */
        sllq1 += 2; /* by columns */
        dllq1 += 2; /* by columns */
        c++;
        if (c <= bandwidth)
                response = HIGH_RESPONSE*GAIN;
        else
                response = LOW_RESPONSE*GAIN;
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sulq2,dulq2);
        CMPLX_UPDATE(sllq1,dllq1);
        }
/*
Now process the elements in the first two columns of the
bottom half of the t_fft2d result.
Initialize the pointers and variables.
*/
    sllq1 = SRC + nyquist*YADDRINC;
    dllq1 = DEST + nyquist*YADDRINC;
    ysq = (nyquist-1)*(nyquist-1);
    c = 0;
    loop (nyquist - 1)
        {
/*
Bump pointers appropriately, by rows
*/
        sllq1 += YADDRINC;
        dllq1 += YADDRINC;
        c++;
/*
As all elements in these columns have the x harmonic equal
to the nyquist value, this is also checked before
filtering
*/
```

```
        if ((c <= bandwidth)&&(nyquist<=bandwidth))
                response = HIGH_RESPONSE*GAIN;
        else
                response = LOW_RESPONSE*GAIN;
        CMPLX_UPDATE(s11q1,d11q1);
        }
/*
Finally process real only elements
*/
/*
DC term: this may be adjusted; in most cases it is
simply set to HIGH RESPONSE * GAIN
*/
    response = HIGH_RESPONSE*GAIN;
    set_ac(AC_LDALL,SRC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST);
    write_ac(aslong(real));
/*
Nyquist,0
*/
    if (nyquist <= bandwidth)
        response = HIGH_RESPONSE*GAIN;
    else
        response = LOW_RESPONSE*GAIN;
    set_ac(AC_LDALL,SRC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+1);
    write_ac(aslong(real));
/*
0,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC);
    write_ac(aslong(real));
/*
Nyquist,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC+1);
```

```
    write_ac(aslong(real));
/*
Reset addressing mode
*/
    output(WR_AM,OldAM);
/*
As the t_ifft2d routine will write over the data it is working
on, the filter results were copied to a separate location to
save for viewing the overall shape.
*/
    t_copy(SAVEFILTER_X,SAVEFILTER_Y,512,0,SIZE,SIZE);
    t_ifft2d(SAVEFILTER_X,SAVEFILTER_Y,RESULT_X,RESULT_Y,
             SIZE,SIZE,WIDTH,OFFSET);
    return(IPL_SUCCESS);
}
```

### *C.2.3 CircIdealLPF.tc*

```
/*****************************  *****************************/
/* FILE : CircIdealLPF.tc, located in the                    */
/*      ~robern/standalone_taac directory of Louvre          */
/* DATE : 31 August 1990                                     */
/* VERSION : 1.0                                             */
/* AUTHOR : R. Oberndorf                                     */
/* DESCRIPTION :  This program performs the 2DFFT of an image,  */
/*      filters the result with a circular low-pass filter, and */
/*      does an inverse 2DFFT on the filtered data.  The guts of*/
/*      the filtering process is based on the lowpass.tc program*/
/*      located in the /taac/src/taaclib/image directory of the */
/*      Sun-4 Workstations.  The filters available in the TAAC-1*/
/*      library are Butterworth type filters;  it was desired to*/
/*      have a more ideal filter, hence this program.        */
/* NOTE :  The section of Appendix A which deals with t_fft2d   */
/*      contains a figure which provides the data layout of the */
/*      t_fft2d result.  That figure, used in conjunction with  */
/*      the comments in this program, should provide sufficient */
/*      detail in understanding the "why's and wherefore's" of  */
/*      this program.                                        */
/* FUNCTIONS :                                               */
/*      CMPLX_UPDATE(src, dest) - performs the actual filtering */
/*              process on the DRAM data.  Takes the data stored*/
/*              at the source (src) address, multiplies it by   */
/*              the filter response, and stores the result at   */
/*              the destination (dest) address                  */
/*      set_ac(AC_LDALL, src) - sets the Address Count Register */
/*              (AC) to the source (src) address                */
/*      read_ac() - reads data from the address in the AC       */
/*      write_ac(response) - writes value of response to the    */
/*              address stored in the AC                        */
/*      t_fftinit() - initializes the TAAC-1 look-up table with */
/*              FFT weights                                     */
/*      t_fft2d() - performs 2DFFT on the SRC integer data,     */
/*              placing the floating-point result at the DEST   */
/*      t_copy() - copies data from one area of the DRAM to     */
/*              another.  Saves original data.                  */
/*      t_ifft2d() - performs the inverse 2DFFT process on the  */
/*              SRC floating-point data, and stores the integer */
/*              result at DEST.                                 */
/* HISTORY:  This code was originally designed as part of a     */
```

```
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.                                                 */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,     */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;        */
/*      TAAC-1 Application Accelerator: Software Reference      */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/************************************************************/
/*  add libraries necessary to run various subroutines */
#include </taac/include/taac1/t_ipl.h>
#include </taac/include/taac1/builtin.h>
#include </taac/include/taac1/t_graphics.h>

/*  image size in pixels */
#define SIZE 128
/*  multiplied by GAIN to get the filter extremes  */
#define LOW_RESPONSE 0
#define HIGH_RESPONSE 1
/*  acts as contrast variance  */
#define GAIN .04
/*  upper bandwidth factor: cutoff harmonic = BW * SIZE / 2  */
#define BW .125
/*  x,y location of fft results   */
#define SAVEFFT_X 0
#define SAVEFFT_Y 512
/*  x,y location of low pass filter results  */
#define SAVEFILTER_X 128
#define SAVEFILTER_Y 512
/*  x,y location of inverse-fft results   */
#define RESULT_X 128
#define RESULT_Y 0
/*  RGB determination  */
#define WIDTH 16
#define OFFSET 0

/*
The following subroutine multiplies the filter response by the
fft values in the source location and stores the result in
the destination addresses.
*/
#define CMPLX_UPDATE(src,dest) \
    { \
    set_ac(AC_LDALL,src); \
```

```
    real = response*asfloat(read_ac()); \
    upd_ac(INC_COL); \
    imag = response*asfloat(read_ac()); \
    set_ac(AC_LDALL,dest); \
    write_ac(aslong(real)); \
    upd_ac(INC_COL); \
    write_ac(aslong(imag)); \
    }

main()
{
    register RD int     sulq1, sulq2, sllq1, sllq2,
                        dulq1, dulq2, dllq1, dllq2,
                        sp1, sp2, sp3, sp4,
                        dp1, dp2, dp3, dp4;

    register RC int     DiagInc1, DiagInc2,
                        ysq;

    register float      dist, A1, A2, bandwidth,
                        response, real, imag;

    register int        i, r, c,
                        rows, cols;
    int                 SRC, DEST,nyquist,
                        OldAM;

/*
Initialize the TAAC-1 look-up table with FFT weights, and perform
the 2DFFT of the original image
*/
    t_fftinit();
    t_fft2d(0,0,SAVEFFT_X, SAVEFFT_Y,SIZE,SIZE,WIDTH,OFFSET);

/*
Save AM register mode
*/
    OldAM = input(RD_AM);

/*
Set AM register for 2d addressing mode
*/
```

```
    output(WR_AM,TA_AM2D);

/*
Create 2d addresses for source(s) and destination
*/
    SRC = ((SAVEFFT_Y << 16) | SAVEFFT_X);
    DEST = ((SAVEFILTER_Y << 16) | SAVEFILTER_X);

/*
Compute filter transfer function parameters
*/
    nyquist = SIZE/2;
    bandwidth = BW*nyquist;

/*
The filtering method which follows is based on the figure shown
in Appendix A (in the section dealing with t_fft2d) which gives
the layout of the t_fft2d result.  Much of the method used here
is based on the symmetry between the upper and lower halves of
the t_fft2d result.
*/
/*
First process elements along diagonal lines of symmetry going
from the top left to the bottom right in the top half of the
result, and from the bottom left to top right for the bottom
half.
Initialize variables and pointers: the DiagIncs will increment
the pointers diagonally
*/
    DiagInc1 = YADDRINC + 2;
    DiagInc2 = YADDRINC - 2;
/*
Starting points for the upper half source (sulq)
and destination (dulq):
*/
    sulq1 = SRC;
    dulq1 = DEST;
/*
Starting points for the lower half source (sllq)
and destination (dllq):
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC;
```

```
    dllq1 = DEST + (SIZE-1)*YADDRINC;

    rows = nyquist - 1;
    r = 0;
    loop(rows)
        {
/*
Increment upper left quadrant pointers (moving diagonally top
left to bottom right)
*/
        sulq1 += DiagInc1;
        dulq1 += DiagInc1;
/*
Decrement lower left quadrant pointers (moving diagonally bottom
left to top right
*/
        sllq1 -= DiagInc2;
        dllq1 -= DiagInc2;
        r++;
/*
Along the diagonals, Pythagorean equation used to filter out
the higher frequency components
*/
        if ((2*r*r) <= (bandwidth*bandwidth))
                response = HIGH_RESPONSE*GAIN;
        else
                response = LOW_RESPONSE*GAIN;
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sllq1,dllq1);
        }

/*
Now process elements that are symmetric about diagonals
just done.  At the end of this section, the entire t_fft2d
result will have been filtered EXCEPT the top row of components,
the bottom row, and the first two columns on the left.
Initialize variables and pointers:
*/
    r = 0;
    rows = nyquist - 2;
    cols = nyquist - 1;
/*
```

```
Two pointers are used for each half - each of the two works on
one a separate side of the diagonal.  First the top half pointers
are initialized, sulq1 and dulq1 working on the right side of the
diagonal, sulq2 and dulq2 on the left side:
*/
    sulq1 = SRC + 2;
    dulq1 = DEST + 2;
    sulq2 = SRC + YADDRINC;
    dulq2 = DEST + YADDRINC;
/*
Then the bottom half pointers are initialized, sllq1 and dllq1
working on the right side of the bottom diagonal, sllq2 and dllq2
on the left side
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC + 2
    dllq1 = DEST + (SIZE-1)*YADDRINC + 2;
    sllq2 = SRC + (SIZE-2)*YADDRINC;
    dllq2 = DEST + (SIZE-2)*YADDRINC;

    loop(rows)
        {
        r++;
        ysq = r*r;
        cols--;
        c = r;
/*
Update upper half pointers downward along diagonal
*/
        sp1 = sulq1 += DiagInc1;
        dp1 = dulq1 += DiagInc1;
        sp2 = sulq2 += DiagInc1;
        dp2 = dulq2 += DiagInc1;
/*
Update lower half pointers upward along diagonal
*/
        sp3 = sllq1 -= DiagInc2;
        dp3 = dllq1 -= DiagInc2;
        sp4 = sllq2 -= DiagInc2;
        dp4 = dllq2 -= DiagInc2;

        loop (cols)
            {
```

```
            c++;
/*
Again, use Euclidean distance to determine if the component
should or should not be filtered out
*/
            dist = ysq + c*c;
            if (dist <= (bandwidth*bandwidth))
                response = HIGH_RESPONSE*GAIN;
            else
                response = LOW_RESPONSE*GAIN;
            CMPLX_UPDATE(sp1,dp1);
            CMPLX_UPDATE(sp2,dp2);
            CMPLX_UPDATE(sp3,dp3);
            CMPLX_UPDATE(sp4,dp4);
/*
Increment upper half, right side pointer by columns
*/
            sp1 += 2; dp1 += 2;
/*
Increment upper half, left side pointer by rows
*/
            sp2 += YADDRINC; dp2 += YADDRINC;
/*
Increment lower half, right side pointer by columns
*/
            sp3 += 2; dp3 += 2;
/*
Decrement lower half, left side pointer by rows
*/
            sp4 -= YADDRINC; dp4 -= YADDRINC;
            }
        }
/*
Next, the top row of the result, the bottom row, and the upper
half left two columns will be filtered.
Initialize pointers and variables: sulq1 works on the top row,
sulq2 on the left two columns, and sllq1 on the bottom row.
*/
    sulq1 = sulq2 = SRC;
    dulq1 = dulq2 = DEST;
    sllq1 = SRC + (SIZE-1)*YADDRINC;
    dllq1 = DEST + (SIZE-1)*YADDRINC;
```

```
    c = 0;
    loop (nyquist - 1)
        {
/*
Move sulq1 and sllq1 to the right, and sulq2 downward
*/
        sulq1 += 2; /* by columns */
        dulq1 += 2; /* by columns */
        sulq2 += YADDRINC; /* by rows */
        dulq2 += YADDRINC; /* by rows */
        sllq1 += 2; /* by columns */
        dllq1 += 2; /* by columns */
        c++;
        if (c <= bandwidth)
                response = HIGH_RESPONSE*GAIN;
        else
                response = LOW_RESPONSE*GAIN;
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sulq2,dulq2);
        CMPLX_UPDATE(sllq1,dllq1);
        }
/*
Now process the elements in the first two columns of the
bottom half of the t_fft2d result.
Initialize the pointers and variables.
*/
    sllq1 = SRC + nyquist*YADDRINC;
    dllq1 = DEST + nyquist*YADDRINC;
    ysq = (nyquist-1)*(nyquist-1);
    c = 0;
    loop (nyquist - 1)
        {
/*
Bump pointers appropriately, by rows
*/
        sllq1 += YADDRINC;
        dllq1 += YADDRINC;
        c++;
/*
As all elements in these columns have the x harmonic equal
to the nyquist value, this is also checked before
filtering
```

```
*/
        if ((c <= bandwidth)&&(nyquist<=bandwidth))
                response = HIGH_RESPONSE*GAIN;
        else
                response = LOW_RESPONSE*GAIN;
        CMPLX_UPDATE(sllq1,dllq1);
        }
/*
Finally process real only elements
*/
/*
DC term: this may be adjusted; in most cases it is
simply set to HIGH RESPONSE * GAIN
*/
    response = HIGH_RESPONSE*GAIN;
    set_ac(AC_LDALL,SRC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST);
    write_ac(aslong(real));
/*
Nyquist,0
*/
    if (nyquist <= bandwidth)
        response = HIGH_RESPONSE*GAIN;
    else
        response = LOW_RESPONSE*GAIN;
    set_ac(AC_LDALL,SRC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+1);
    write_ac(aslong(real));
/*
0,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC);
    write_ac(aslong(real));
/*
Nyquist,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC+1);
    real = response*asfloat(read_ac());
```

```
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC+1);
    write_ac(aslong(real));
/*
Reset addressing mode
*/
    output(WR_AM,OldAM);
/*
As the t_ifft2d routine will write over the data it is working
on, the filter results were copied to a separate location to
save for viewing the overall shape.
*/
    t_copy(SAVEFILTER_X,SAVEFILTER_Y,256,512,SIZE,SIZE);
    t_ifft2d(SAVEFILTER_X,SAVEFILTER_Y,RESULT_X,RESULT_Y,
          SIZE,SIZE,WIDTH,OFFSET);

    return(IPL_SUCCESS);
}
```

### C.2.4 *Makefile (stand-alone)*

```
/*****************************************************************/
/* FILE : Makefile (stand-alone), located in the                 */
/*      ~robern/standalone_taac directory of Louvre              */
/* DATE : 31 August 1990                                         */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  Directs compilation and linkage specifically   */
/*      for stand-alone TAAC-1 programs, written for this thesis*/
/*      effort.  Rather than constantly having to change file    */
/*      names within this Makefile whenever new programs were to*/
/*      be run, a boilerplate file (a1.tc) was used into which   */
/*      a program was first copied and then compiled and linked */
/*      using this Makefile.                                     */
/* FUNCTIONS :                                                   */
/*      tacc (implicit) - C compiler for TAAC-1 program          */
/* HISTORY:  This code was originally designed as part of a      */
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.  It was based on the Makexample file for the    */
/*      programs db.tc, dbchan.tc, et al, located in the         */
/*      /taac/tutorial directory of Louvre.                      */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,      */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;         */
/*      TAAC-1 Application Accelerator: Software Reference       */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/*****************************************************************/
# makefile for stand-alone TAAC-1 programs

# include the predefined rules for building TAAC-1 programs
include /usr/include/taac1/makerules

# define the constants
TCFLAGS = -c -fsingle           # flags for tacc
TLFLAGS =                       # flags for talink
TLIBS = -ltaac1                 # libraries for TAAC-1 program
TOBJS = a1.obj                  # object files for TAAC-1 program

# compile (implicit) and link TAAC-1 program (TALINK is defined in makerules)
# makerules also describes how to make a .obj file from a .tc file
a1: a1.abs

a1.abs: $(TOBJS)
```

```
$(TALINK) $(TLFLAGS) $(TOBJS) -g -o a1.abs $(TLIBS)
```

### *C.2.5 HumanGauss.tc*

```
/*****************************************************************/
/* FILE : HumanGauss.tc, located in the ~robern/standalone_taac */
/*      directory of Louvre                                      */
/* DATE : 09 October 1990                                        */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  This program performs the 2DFFT of an image,  */
/*      filters the result with a Gabor-type low-pass filter,    */
/*      does an inverse 2DFFT on the filtered data.  The guts of*/
/*      the filtering process is based on the lowpass.tc program*/
/*      located in the /taac/src/taaclib/image directory of the */
/*      Sun-4 Workstations.  The filters available in the TAAC-1*/
/*      library are Butterworth type filters;  it was desired to*/
/*      have a more biologically-driven filter, hence this       */
/*      program.                                                 */
/* NOTE :  The section of Appendix A which deals with t_fft2d   */
/*      contains a figure which provides the data layout of the */
/*      t_fft2d result.  That figure, used in conjunction with  */
/*      the comments in this program, should provide sufficient */
/*      detail in understanding the "why's and wherefore's" of  */
/*      this program.                                            */
/* FUNCTIONS :                                                   */
/*      CMPLX_UPDATE(src, dest) - performs the actual filtering */
/*              process on the DRAM data.  Takes the data stored*/
/*              at the source (src) address, multiplies it by   */
/*              the filter response, and stores the result at   */
/*              the destination (dest) address                  */
/*      set_ac(AC_LDALL, src) - sets the Address Count Register */
/*              (AC) to the source (src) address                */
/*      read_ac() - reads data from the address in the AC       */
/*      write_ac(response) - writes value of response to the    */
/*              address stored in the AC                        */
/*      t_fftinit() - initializes the TAAC-1 look-up table with */
/*              FFT weights                                     */
/*      t_fft2d() - performs 2DFFT on the SRC integer data,     */
/*              placing the floating-point result at the DEST   */
/*      t_copy() - copies data from one area of the DRAM to     */
/*              another.  Saves original data.                  */
/*      t_ifft2d() - performs the inverse 2DFFT process on the  */
/*              SRC floating-point data, and stores the integer */
/*              result at DEST.                                 */
```

```
/*      t_pow() - raises the first variable to the power        */
/*              represented by the second variable.             */
/*      t_sqrt() - computes the square root of the variable.    */
/*      t_exp() - computes e to the power represented by the    */
/*              given variable.                                 */
/* HISTORY:  This code was originally designed as part of a     */
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.                                                */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,     */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;        */
/*      TAAC-1 Application Accelerator: Software Reference      */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/***************************************************************/
/*  add libraries necessary to run various subroutines */
#include </taac/include/taac1/t_ipl.h>
#include </taac/include/taac1/builtin.h>
#include </taac/include/taac1/t_graphics.h>
#include </taac/include/taac1/t_math.h>

/*  image size in pixels */
#define SIZE 128
/*  multiplied by GAIN to get the filter extremes  */
#define LOW_RESPONSE 0
#define HIGH_RESPONSE 1
/*  acts as contrast variance  */
#define GAIN 0.04
/*  serve as reference points when filtering at nyquist value  */
#define UPPER_BW 16
#define LOWER_BW 0
/*  the number of gaussians used to represent the MTF  */
#define NUMB_OF_GAUSS 4
/*  x,y location of fft results   */
#define SAVEFFT_X 0
#define SAVEFFT_Y 512
/*  x,y location of low pass filter results  */
#define SAVEFILTER_X 128
#define SAVEFILTER_Y 512
/*  x,y location of inverse-fft results   */
#define RESULT_X 256
#define RESULT_Y 0
/*  RGB determination  */
#define WIDTH 16
```

```
#define OFFSET 0

/*
The following subroutine multiplies the filter response by the
fft values in the source location and stores the result in
the destination addresses.
*/
#define CMPLX_UPDATE(src,dest) \
            { \
            set_ac(AC_LDALL,src); \
            real = response*asfloat(read_ac()); \
            upd_ac(INC_COL); \
            imag = response*asfloat(read_ac()); \
            set_ac(AC_LDALL,dest); \
            write_ac(aslong(real)); \
            upd_ac(INC_COL); \
            write_ac(aslong(imag)); \
            }

main()
{
    register RD int     sulq1, sulq2, sllq1, sllq2,
                        dulq1, dulq2, dllq1, dllq2,
                        sp1, sp2, sp3, sp4,
                        dp1, dp2, dp3, dp4;

    register RC int     DiagInc1, DiagInc2, ysq;

    register float      dist, A1, A2, squareroot, sigma[5],
                        exponent, response, real, imag;

    register int        i, r, c, rows, cols;

    int                 SRC, DEST, nyquist, j, OldAM;

/*
Initialize the TAAC-1 look-up table with FFT weights, and perform
the 2DFFT of the original image
*/
    t_fftinit();
    t_fft2d(0,0,SAVEFFT_X, SAVEFFT_Y,SIZE,SIZE,WIDTH,OFFSET);
```

```
/*
Save AM register mode
*/
    OldAM = input(RD_AM);

/*
Set AM register for 2d addressing mode
*/
    output(WR_AM,TA_AM2D);

/*
Create 2d addresses for source(s) and destination
*/
    SRC = ((SAVEFFT_Y << 16) | SAVEFFT_X);
    DEST = ((SAVEFILTER_Y << 16) | SAVEFILTER_X);

/*
Compute filter transfer function parameters
*/
    nyquist = SIZE/2;
    for(j=1;j<=NUMB_OF_GAUSS;j++)
        sigma[j]=(float)(t_pow(2.,j)/3);

/*
The filtering method which follows is based on the figure shown
in Appendix A (in the section dealing with t_fft2d) which gives
the layout of the t_fft2d result.  Much of the method used here
is based on the symmetry between the upper and lower halves of
the t_fft2d result.
*/
/*
First process elements along diagonal lines of symmetry going
from the top left to the bottom right in the top half of the
result, and from the bottom left to top right for the bottom
half.
Initialize variables and pointers: the DiagIncs will increment
the pointers diagonally
*/
    DiagInc1 = YADDRINC + 2;
    DiagInc2 = YADDRINC - 2;
/*
Starting points for the upper half source (sulq)
```

```
and destination (dulq):
*/
    sulq1 = SRC;
    dulq1 = DEST;
/*
Starting points for the lower half source (sllq)
and destination (dllq):
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC;
    dllq1 = DEST + (SIZE-1)*YADDRINC;

    rows = nyquist - 1;
    r = 0;
    loop(rows) {
/*
Increment upper left quadrant pointers (moving diagonally top
left to bottom right)
*/
        sulq1 += DiagInc1;
        dulq1 += DiagInc1;
/*
Decrement lower left quadrant pointers (moving diagonally bottom
left to top right
*/
        sllq1 -= DiagInc2;
        dllq1 -= DiagInc2;
        r++;
/*
The amount of filtering is based on the total response of
the gaussians at the point in question
*/
        dist = 2*r*r;
        squareroot = t_sqrt(dist);
        response = 0;
        for(j=0;j<NUMB_OF_GAUSS;j++) {
            exponent = 0.-(((squareroot-t_ipow(2.,j))*
                       (squareroot-t_ipow(2.,j)))/
                        (2*sigma[j]*sigma[j]));
            response += t_exp(exponent)*GAIN;
        }
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sllq1,dllq1);
```

```
    }

/*
Now process elements that are symmetric about diagonals
just done.  At the end of this section, the entire t_fft2d
result will have been filtered EXCEPT the top row of components,
the bottom row, and the first two columns on the left.
Initialize variables and pointers:
*/
    r = 0;
    rows = nyquist - 2;
    cols = nyquist - 1;

/*
Two pointers are used for each half - each of the two works on
one a separate side of the diagonal.  First the top half pointers
are initialized, sulq1 and dulq1 working on the right side of the
diagonal, sulq2 and dulq2 on the left side:
*/
    sulq1 = SRC + 2;
    dulq1 = DEST + 2;
    sulq2 = SRC + YADDRINC;
    dulq2 = DEST + YADDRINC;
/*
Then the bottom half pointers are initialized, sllq1 and dllq1
working on the right side of the bottom diagonal, sllq2 and dllq2
on the left side
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC + 2;
    dllq1 = DEST + (SIZE-1)*YADDRINC + 2;
    sllq2 = SRC + (SIZE-2)*YADDRINC;
    dllq2 = DEST + (SIZE-2)*YADDRINC;

    loop(rows) {
        r++;
        ysq = r*r;
        cols--;
        c = r;

/*
Update upper half pointers downward along diagonal
*/
```

```
        sp1 = sulq1 += DiagInc1;
        dp1 = dulq1 += DiagInc1;
        sp2 = sulq2 += DiagInc1;
        dp2 = dulq2 += DiagInc1;
/*
Update lower half pointers upward along diagonal
*/
        sp3 = sllq1 -= DiagInc2;
        dp3 = dllq1 -= DiagInc2;
        sp4 = sllq2 -= DiagInc2;
        dp4 = dllq2 -= DiagInc2;

        loop (cols) {
            c++;
/*
Again, the amount of filtering is based on the total response of
the gaussians at the point in question
*/
            dist = ysq + c*c;
            squareroot = t_sqrt(dist);
            response = 0;
            for(j=0;j<NUMB_OF_GAUSS;j++) {
                exponent = 0.-(((squareroot-t_ipow(2.,j))*
                           (squareroot-t_ipow(2.,j)))/
                            (2*sigma[j]*sigma[j]));
                response += t_exp(exponent)*GAIN;
            }
            CMPLX_UPDATE(sp1,dp1);
            CMPLX_UPDATE(sp2,dp2);
            CMPLX_UPDATE(sp3,dp3);
            CMPLX_UPDATE(sp4,dp4);
/*
Increment upper half, right side pointer by columns
*/
            sp1 += 2; dp1 += 2;
/*
Increment upper half, left side pointer by rows
*/
            sp2 += YADDRINC; dp2 += YADDRINC;
/*
Increment lower half, right side pointer by columns
*/
```

```
            sp3 += 2; dp3 += 2;
/*
Decrement lower half, left side pointer by rows
*/
            sp4 -= YADDRINC; dp4 -= YADDRINC;
        }
    }
/*
Next, the top row of the result, the bottom row, and the upper
half left two columns will be filtered.
Initialize pointers and variables: sulq1 works on the top row,
sulq2 on the left two columns, and sllq1 on the bottom row.
*/
    sulq1 = sulq2 = SRC;
    dulq1 = dulq2 = DEST;
    sllq1 = SRC + (SIZE-1)*YADDRINC;
    dllq1 = DEST + (SIZE-1)*YADDRINC;
    c = 0;
    loop (nyquist - 1) {
/*
Move sulq1 and sllq1 to the right, and sulq2 downward
*/
        sulq1 += 2; /* by columns */
        dulq1 += 2; /* by columns */
        sulq2 += YADDRINC; /* by rows */
        dulq2 += YADDRINC; /* by rows */
        sllq1 += 2; /* by columns */
        dllq1 += 2; /* by columns */
        c++;
        squareroot=c;
        response = 0;
        for(j=0;j<NUMB_OF_GAUSS;j++) {
            exponent = 0.-(((squareroot-t_ipow(2.,j))*
                       (squareroot-t_ipow(2.,j)))/
                        (2*sigma[j]*sigma[j]));
            response += t_exp(exponent)*GAIN;
        }
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sulq2,dulq2);
        CMPLX_UPDATE(sllq1,dllq1);
    }
/*
```

```
Now process the elements in the first two columns of the
bottom half of the t_fft2d result.
Initialize the pointers and variables.
*/
    sllq1 = SRC + nyquist*YADDRINC;
    dllq1 = DEST + nyquist*YADDRINC;
    ysq = (nyquist-1)*(nyquist-1);
    c = 0;
    if(nyquist<=UPPER_BW)
        loop (nyquist - 1) {
/*
Bump pointers appropriately, by rows
*/
            sllq1 += YADDRINC;
            dllq1 += YADDRINC;
            c++;
            squareroot=c;
            response = 0;
            for(j=0;j<NUMB_OF_GAUSS;j++) {
                exponent = 0.-(((squareroot-t_ipow(2.,j))*
                            (squareroot-t_ipow(2.,j)))/
                              (2*sigma[j]*sigma[j]));
                response += t_exp(exponent)*GAIN;
            }
            CMPLX_UPDATE(sllq1,dllq1);
        }
    else
        loop (nyquist - 1) {
/*
Bump pointers appropriately, by rows
*/
            sllq1 += YADDRINC;
            dllq1 += YADDRINC;
            response=0;
            CMPLX_UPDATE(sllq1,dllq1);
        }
/*
Finally process real only elements beginning with the DC term
*/
    for(j=0;j<NUMB_OF_GAUSS;j++) {
        exponent = 0.-((t_ipow(2.,j)*t_ipow(2.,j))/(2*sigma[j]*sigma[j]));
        response += t_exp(exponent)*GAIN;
```

```
    }
    set_ac(AC_LDALL,SRC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST);
    write_ac(aslong(real));
/*
Nyquist,0
*/
    if ((nyquist <= UPPER_BW)&&(nyquist>=LOWER_BW))
        response = HIGH_RESPONSE*GAIN;
    else
        response = LOW_RESPONSE*GAIN;
    set_ac(AC_LDALL,SRC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+1);
    write_ac(aslong(real));
/*
0,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC);
    write_ac(aslong(real));
/*
Nyquist,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC+1);
    write_ac(aslong(real));
/*
Reset addressing mode
*/
    output(WR_AM,OldAM);
/*
As the t_ifft2d routine will write over the data it is working
on, the filter results were copied to a separate location to
save for viewing the overall shape.
*/
    t_copy(SAVEFILTER_X,SAVEFILTER_Y,256,512,SIZE,SIZE);
    t_ifft2d(SAVEFILTER_X,SAVEFILTER_Y,RESULT_X,RESULT_Y,
        SIZE,SIZE,WIDTH,OFFSET);
```

```
    return(IPL_SUCCESS);
}
```

### *C.2.6 FineTunedGaussian.tc*

```
/*****************************************************************/
/* FILE : FineTunedGaussian.tc, located in the                   */
/*      ~robern/standalone_taac directory of Louvre              */
/* DATE : 09 October 1990                                        */
/* VERSION : 1.0                                                 */
/* AUTHOR : R. Oberndorf                                         */
/* DESCRIPTION :  This program performs the 2DFFT of an image,   */
/*      filters the result with a Gabor-type low-pass filter,    */
/*      does an inverse 2DFFT on the filtered data.  The guts of*/
/*      the filtering process is based on the lowpass.tc program*/
/*      located in the /taac/src/taaclib/image directory of the */
/*      Sun-4 Workstations.  The filters available in the TAAC-1*/
/*      library are Butterworth type filters;  it was desired to*/
/*      have a more biologically-driven filter, hence this       */
/*      program.                                                 */
/* NOTE :  The section of Appendix A which deals with t_fft2d    */
/*      contains a figure which provides the data layout of the */
/*      t_fft2d result.  That figure, used in conjunction with   */
/*      the comments in this program, should provide sufficient */
/*      detail in understanding the "why's and wherefore's" of   */
/*      this program.                                            */
/* FUNCTIONS :                                                   */
/*      CMPLX_UPDATE(src, dest) - performs the actual filtering */
/*              process on the DRAM data.  Takes the data stored*/
/*              at the source (src) address, multiplies it by    */
/*              the filter response, and stores the result at    */
/*              the destination (dest) address                   */
/*      set_ac(AC_LDALL, src) - sets the Address Count Register */
/*              (AC) to the source (src) address                 */
/*      read_ac() - reads data from the address in the AC        */
/*      write_ac(response) - writes value of response to the     */
/*              address stored in the AC                         */
/*      t_fftinit() - initializes the TAAC-1 look-up table with */
/*              FFT weights                                      */
/*      t_fft2d() - performs 2DFFT on the SRC integer data,      */
/*              placing the floating-point result at the DEST    */
/*      t_copy() - copies data from one area of the DRAM to      */
/*              another.  Saves original data.                   */
/*      t_ifft2d() - performs the inverse 2DFFT process on the   */
/*              SRC floating-point data, and stores the integer */
/*              result at DEST.                                  */
```

```
/*      t_pow() - raises the first variable to the power        */
/*              represented by the second variable.             */
/*      t_sqrt() - computes the square root of the variable.    */
/*      t_exp() - computes e to the power represented by the    */
/*              given variable.                                 */
/* HISTORY:  This code was originally designed as part of a     */
/*      thesis effort, Analysis of Visual Illusions Using Gabor */
/*      Filters.                                                */
/* REFERENCES:  TAAC-1 Application Accelerator: User Guide,     */
/*      Sun Microsystems Inc., Part Number: 800-2177-11;        */
/*      TAAC-1 Application Accelerator: Software Reference      */
/*      Manual, Sun Microsystems Inc., Part Number: 800-3202-11 */
/***************************************************************/
/*  add libraries necessary to run various subroutines */
#include </taac/include/taac1/t_ipl.h>
#include </taac/include/taac1/builtin.h>
#include </taac/include/taac1/t_graphics.h>
#include </taac/include/taac1/t_math.h>

/*  image size in pixels */
#define SIZE 128
/*  multiplied by GAIN to get the filter extremes  */
#define LOW_RESPONSE 0
#define HIGH_RESPONSE 1
/*  acts as contrast variance  */
#define GAIN 0.04
/*
the extent of the low-pass filtering, gaussians at each
harmonic out to the upper bandwidth
*/
#define UPPER_BW 32
#define UPPER_SQUARED UPPER_BW*UPPER_BW
#define LOWER_BW 0
/*  a uniform gaussian spread is used  */
#define SIGMA .5
/*  x,y location of fft results   */
#define SAVEFFT_X 768
#define SAVEFFT_Y 512
/*  x,y location of low pass filter results  */
#define SAVEFILTER_X 768
#define SAVEFILTER_Y 768
/*  x,y location of inverse-fft results   */
```

```
#define RESULT_X 512
#define RESULT_Y 0
/*  RGB determination  */
#define WIDTH 16
#define OFFSET 0
/*
The following subroutine multiplies the filter response by the
fft values in the source location and stores the result in
the destination addresses.
*/
#define CMPLX_UPDATE(src,dest) \
            { \
            set_ac(AC_LDALL,src); \
            real = response*asfloat(read_ac()); \
            upd_ac(INC_COL); \
            imag = response*asfloat(read_ac()); \
            set_ac(AC_LDALL,dest); \
            write_ac(aslong(real)); \
            upd_ac(INC_COL); \
            write_ac(aslong(imag)); \
            }

main()
{
    register RD int     sulq1, sulq2, sllq1, sllq2,
                        dulq1, dulq2, dllq1, dllq2,
                        sp1, sp2, sp3, sp4,
                        dp1, dp2, dp3, dp4;

    register RC int     DiagInc1, DiagInc2, ysq;

    register float      dist, squareroot, exponent,
                response, real, imag;

    register int        i, r, c,
                        rows, cols;
    int                 SRC, DEST, nyquist, mid, OldAM;

/*
Initialize the TAAC-1 look-up table with FFT weights, and perform
the 2DFFT of the original image
*/
```

```
    t_fftinit();
    t_fft2d(0,0,SAVEFFT_X, SAVEFFT_Y,SIZE,SIZE,WIDTH,OFFSET);
/*
Save AM register mode
*/
    OldAM = input(RD_AM);
/*
Set AM register for 2d addressing mode
*/
    output(WR_AM,TA_AM2D);
/*
Create 2d addresses for source(s) and destination
*/
    SRC = ((SAVEFFT_Y << 16) | SAVEFFT_X);
    DEST = ((SAVEFILTER_Y << 16) | SAVEFILTER_X);
/*
Compute filter transfer function parameter
*/
    nyquist = SIZE/2;
/*
The filtering method which follows is based on the figure shown
in Appendix A (in the section dealing with t_fft2d) which gives
the layout of the t_fft2d result.  Much of the method used here
is based on the symmetry between the upper and lower halves of
the t_fft2d result.
*/
/*
First process elements along diagonal lines of symmetry going
from the top left to the bottom right in the top half of the
result, and from the bottom left to top right for the bottom
half.
Initialize variables and pointers: the DiagIncs will increment
the pointers diagonally
*/
    DiagInc1 = YADDRINC + 2;
    DiagInc2 = YADDRINC - 2;
/*
Starting points for the upper half source (sulq)
and destination (dulq):
*/
    sulq1 = SRC;
    dulq1 = DEST;
```

```
/*
Starting points for the lower half source (sllq)
and destination (dllq):
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC;
    dllq1 = DEST + (SIZE-1)*YADDRINC;

    rows = nyquist - 1;
    r = 0;
    loop(rows) {
/*
Increment upper left quadrant pointers (moving diagonally top
left to bottom right)
*/
        sulq1 += DiagInc1;
        dulq1 += DiagInc1;
/*
Decrement lower left quadrant pointers (moving diagonally bottom
left to top right
*/
        sllq1 -= DiagInc2;
        dllq1 -= DiagInc2;
        r++;
/*
The amount of filtering is based on the total response of
the gaussians at the point in question
*/
        dist = 2*r*r;
        squareroot = t_sqrt(dist);
        response=LOW_RESPONSE*GAIN;
        if(dist<=UPPER_SQUARED)
            for(mid=0;mid<=UPPER_BW;mid++) {
                exponent = 0.-(((squareroot-mid)*(squareroot-mid))/
                                (2*SIGMA*SIGMA));
                response += t_exp(exponent)*GAIN;
            }
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sllq1,dllq1);
    }
/*
Now process elements that are symmetric about diagonals
just done.  At the end of this section, the entire t_fft2d
```

```
result will have been filtered EXCEPT the top row of components,
the bottom row, and the first two columns on the left.
Initialize variables and pointers:
*/
    r = 0;
    rows = nyquist - 2;
    cols = nyquist - 1;
/*
Two pointers are used for each half - each of the two works on
one a separate side of the diagonal.  First the top half pointers
are initialized, sulq1 and dulq1 working on the right side of the
diagonal, sulq2 and dulq2 on the left side:
*/
    sulq1 = SRC + 2;
    dulq1 = DEST + 2;
    sulq2 = SRC + YADDRINC;
    dulq2 = DEST + YADDRINC;
/*
Then the bottom half pointers are initialized, sllq1 and dllq1
working on the right side of the bottom diagonal, sllq2 and dllq2
on the left side
*/
    sllq1 = SRC + (SIZE-1)*YADDRINC + 2;
    dllq1 = DEST + (SIZE-1)*YADDRINC + 2;
    sllq2 = SRC + (SIZE-2)*YADDRINC;
    dllq2 = DEST + (SIZE-2)*YADDRINC;

    loop(rows) {
        r++;
        ysq = r*r;
        cols--;
        c = r;
/*
Update upper half pointers downward along diagonal
*/
        sp1 = sulq1 += DiagInc1;
        dp1 = dulq1 += DiagInc1;
        sp2 = sulq2 += DiagInc1;
        dp2 = dulq2 += DiagInc1;
/*
Update lower half pointers upward along diagonal
*/
```

```
        sp3 = sllq1 -= DiagInc2;
        dp3 = dllq1 -= DiagInc2;
        sp4 = sllq2 -= DiagInc2;
        dp4 = dllq2 -= DiagInc2;

        loop (cols) {
            c++;
/*
Again, the amount of filtering is based on the total response of
the gaussians at the point in question
*/
            dist = ysq + c*c;
            squareroot = t_sqrt(dist);
            response=LOW_RESPONSE*GAIN;
            if(dist<=UPPER_SQUARED)
                for(mid=0;mid<=UPPER_BW;mid++) {
                    exponent=0.-(((squareroot-mid)*(squareroot-mid))/
                                (2*SIGMA*SIGMA));
                    response += t_exp(exponent)*GAIN;
                }
            CMPLX_UPDATE(sp1,dp1);
            CMPLX_UPDATE(sp2,dp2);
            CMPLX_UPDATE(sp3,dp3);
    CMPLX_UPDATE(sp4,dp4);
/*
Increment upper half, right side pointer by columns
*/
            sp1 += 2; dp1 += 2;
/*
Increment upper half, left side pointer by rows
*/
            sp2 += YADDRINC; dp2 += YADDRINC;
/*
Increment lower half, right side pointer by columns
*/
            sp3 += 2; dp3 += 2;
/*
Decrement lower half, left side pointer by rows
*/
            sp4 -= YADDRINC; dp4 -= YADDRINC;
        }
    }
```

```
/*
Next, the top row of the result, the bottom row, and the upper
half left two columns will be filtered.
Initialize pointers and variables: sulq1 works on the top row,
sulq2 on the left two columns, and sllq1 on the bottom row.
*/
    sulq1 = sulq2 = SRC;
    dulq1 = dulq2 = DEST;
    sllq1 = SRC + (SIZE-1)*YADDRINC;
    dllq1 = DEST + (SIZE-1)*YADDRINC;
    c = 0;
    loop (nyquist - 1) {
/*
Move sulq1 and sllq1 to the right, and sulq2 downward
*/
        sulq1 += 2; /* by columns */
        dulq1 += 2; /* by columns */
        sulq2 += YADDRINC; /* by rows */
        dulq2 += YADDRINC; /* by rows */
        sllq1 += 2; /* by columns */
        dllq1 += 2; /* by columns */
        c++;
        squareroot=c;
        response=LOW_RESPONSE*GAIN;
        if(squareroot<=UPPER_BW)
            for(mid=0;mid<=UPPER_BW;mid++) {
                exponent = 0.-(((squareroot-mid)*(squareroot-mid))/
                            (2*SIGMA*SIGMA));
                response += t_exp(exponent)*GAIN;
            }
        CMPLX_UPDATE(sulq1,dulq1);
        CMPLX_UPDATE(sulq2,dulq2);
        CMPLX_UPDATE(sllq1,dllq1);
    }
/*
Now process the elements in the first two columns of the
bottom half of the t_fft2d result.
Initialize the pointers and variables.
*/
    sllq1 = SRC + nyquist*YADDRINC;
    dllq1 = DEST + nyquist*YADDRINC;
    ysq = (nyquist-1)*(nyquist-1);
```

```
    c = 0;
    if(nyquist<=UPPER_BW)
        loop (nyquist - 1) {
/*
Bump pointers appropriately, by rows
*/
            sllq1 += YADDRINC;
            dllq1 += YADDRINC;
            c++;
            squareroot=c;
            response=LOW_RESPONSE*GAIN;
            if(squareroot<=UPPER_BW)
                for(mid=0;mid<=UPPER_BW;mid++) {
                    exponent =0.-(((squareroot-mid)*(squareroot-mid))/
                                (2*SIGMA*SIGMA));
                    response += t_exp(exponent)*GAIN;
                }
            CMPLX_UPDATE(sllq1,dllq1);
        }
    else
        loop (nyquist - 1) {
/*
Bump pointers appropriately, by rows
*/
            sllq1 += YADDRINC;
            dllq1 += YADDRINC;
            response=LOW_RESPONSE*GAIN;
            CMPLX_UPDATE(sllq1,dllq1);
}
/*
Finally process real only elements beginning with the DC term
*/
    for(mid=0;mid<=UPPER_BW;mid++) {
        exponent =0.-((mid*mid)/(2*SIGMA*SIGMA));
        response += t_exp(exponent)*GAIN;
    }
    set_ac(AC_LDALL,SRC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST);
    write_ac(aslong(real));
/*
Nyquist,0
```

```
*/
    if ((nyquist <= UPPER_BW)&&(nyquist>=LOWER_BW))
        response = HIGH_RESPONSE*GAIN;
    else
        response = LOW_RESPONSE*GAIN;
    set_ac(AC_LDALL,SRC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+1);
    write_ac(aslong(real));
/*
0,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC);
    write_ac(aslong(real));
/*
Nyquist,Nyquist:  filtering decision made above for Nyq,0
*/
    set_ac(AC_LDALL,SRC+(nyquist)*YADDRINC+1);
    real = response*asfloat(read_ac());
    set_ac(AC_LDALL,DEST+(nyquist)*YADDRINC+1);
    write_ac(aslong(real));
/*
Reset addressing mode
*/
    output(WR_AM,OldAM);
/*
As the t_ifft2d routine will write over the data it is working
on, the filter results were copied to a separate location to
save for viewing the overall shape.
*/
    t_copy(SAVEFILTER_X,SAVEFILTER_Y,384,128,SIZE,SIZE);
    t_ifft2d(SAVEFILTER_X,SAVEFILTER_Y,RESULT_X,RESULT_Y,
           SIZE,SIZE,WIDTH,OFFSET);
   return(IPL_SUCCESS);
}
```

# Appendix D. *Saving/Printing of Images*

Any portion of the TAAC-1 video memory can be saved in order to generate an encapsulated postscript (eps) file for inclusion within a LaTeXdocument by following this procedure:

1. The **taread** host utility may be used to save a portion of the video memory in Image File Format (iff): **taread -x starting_x_location -y starting_y_location -X pixel_width -Y pixel_width > file1.iff.** (Defaults are x=y=0, X=Y=512.)
2. The iff file must now be converted to rle format via: **ifftorle file1.iff file2.rle.**
3. Once the file is in rle form, the last conversion it undergoes is to the eps format: **rle2eps file2 width_in_inches** (Note the .rle extension is not necessary; also, the output will automatically be to file2.eps).
4. To include the file within a LaTeXdocument, certain LaTeXcommands must be used:
   - Add '**epsf**' to the **\documentstyle** line within the same brackets that the type size is stated: **\documentstyle [epsf,12pt]{article}** .
   - If a caption is desired, use the **\begin{figure}** and **\end{figure}** commands surrounding the following line: **\epsffile{file2.eps}** . If no caption is desired, the **epsffile** line may be used by itself.
   - Some adjustment may be necessary using **\hspace** or **\vspace** commands.

Should it be desired to only store the displayed image for the purpose of redisplay at a later time, Step 1 above may be followed at a later time with a **taload** command: **taload -f file1.iff.** Detailed information on **taread** and **taload** can be found in the Utilities chapter of the TAAC-1 User Guide (47), with an abbreviated description in the Utilities section of Appendix A.

## Appendix E. *Frequency Domain Use of Gabor Filters*

In keeping with the biologically-driven theme of this effort, a filter was required which emulated the spectral response of a simple cell in the visual cortex. To make the mathematics much easier, the filtering was to be done in the frequency domain as was the case with the SILPF and CILPF. This allowed for a simple multiplication of Fourier components as follows:

$$FilteredImage = \mathcal{F}(OriginalImage) \times \mathcal{F}(GaborFunction) \tag{9}$$

where the Gabor Function, originally listed in Chapter 2 as Equation 4, is

$$g(x,y) = K \cdot e^{-\frac{1}{2}\left(\frac{x_g^2}{a^2}+\frac{y_g^2}{b^2}\right)} \cdot \cos[-2\pi(U_o x + V_o y) - P]$$

The Fourier Transform of the 2D Gabor Function can be stated as (22:1236)

$$G(u,v) = e^{-iP}e^{-\pi(U_g{}^2a^2+V_g{}^2b^2)} + e^{iP}e^{-\pi(-U_g{}^2a^2-V_g{}^2b^2)} \tag{10}$$

In the spatial domain, the Gabor Function can be thought of as the convolution of a sinusoidal plane wave with an elliptic Gaussian. In the spatial frequency domain, the equation above represents the product of a pair of impulses at a specific frequency by an elliptic Gaussian. These relationships are shown in Figure 36 (22:1235).

As this Gaussian represented the spectral response of a single simple cell, the question became how to combine these Gaussians in order to create a GLPF. A program was written, **humangauss.tc** listed in Appendix C, which performed a summation of Gaussians, each of which was centered about a different frequency. Parameters manipulated through this program were the $\sigma$ (spread) of each Gaussian, the number of Gaussians, and the center frequency of each Gaussian. As an example, Figure 37 provides a 3D view of Gaussians a radial distance away from the origin of 1, 2, 4 and 8 units, with the $\sigma$ of each Gaussian being

$\frac{2}{3}$, $\frac{4}{3}$, $\frac{8}{3}$, and $\frac{16}{3}$ respectively. This example was chosen due to its similarity to the MTF which Ginsburg used in his evaluation (see Figure 12-C).

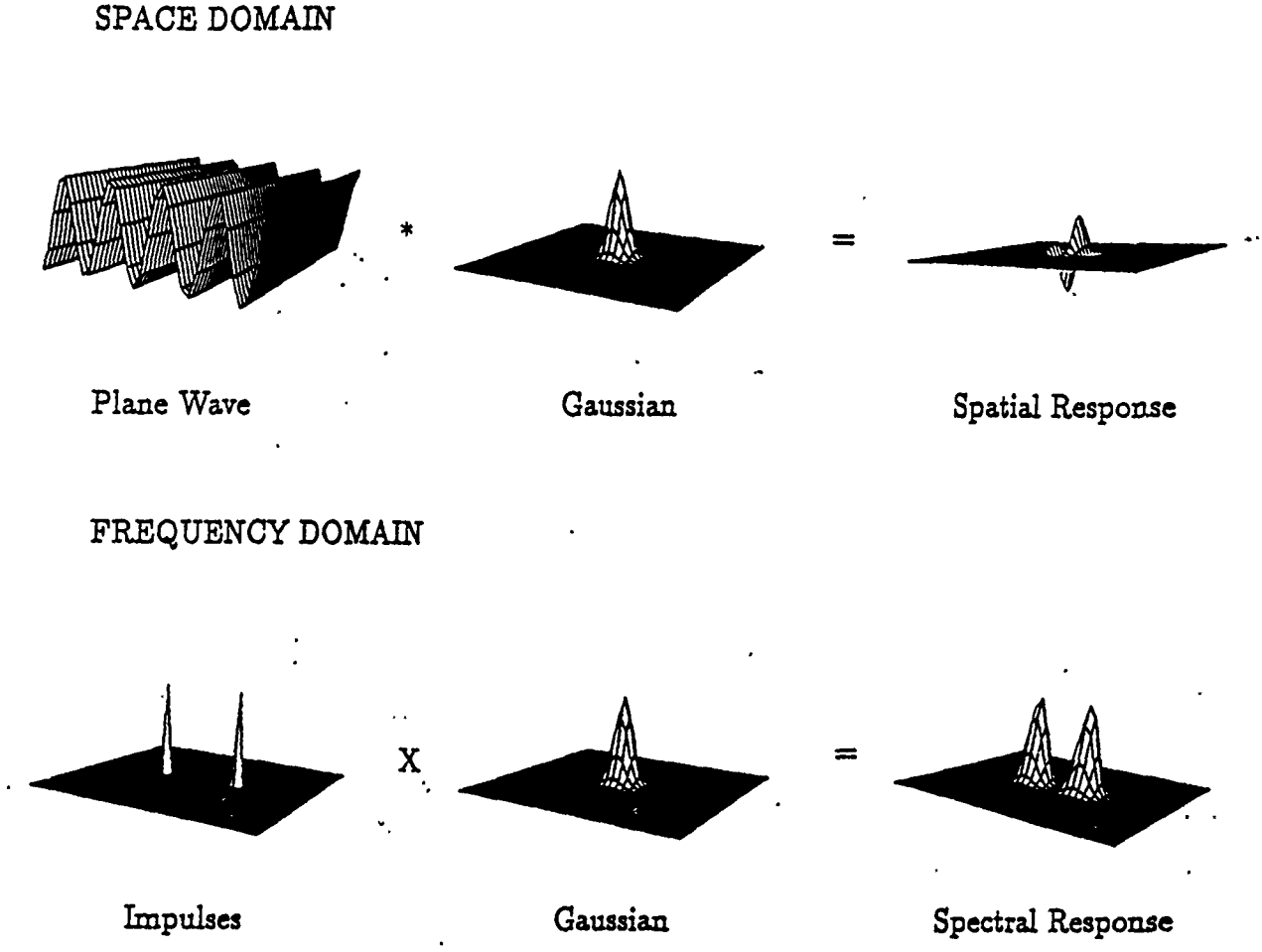



Figure 36 Relationship Between the Spatial and Spatial Frequency Representations of a Gabor Filter

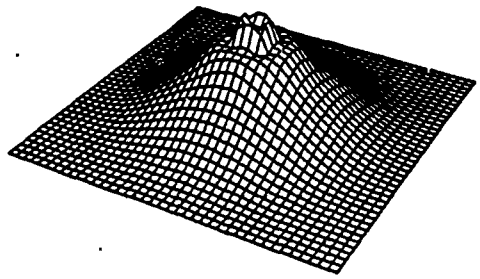


Figure 37. Example of Gabor Low Pass Filter Using a Summation of Gaussians

# *Bibliography*


1. Albrecht, Duane G. and others. "Cortical cells: bar and edge detectors, or spatial frequency filters," *Frontiers of Visual Science*, *207*:88–90 (1981).
2. Andrews, B. W. and D. A. Pollen. "Relationship Between Spatial Frequency Selectivity and Receptive field Profile of Simple Cells," *Journal of Physiology, London*, *287*:163–176 (1979).
3. Blakemore, C. B. and F. W. Campbell. "On the Existence of Neurons in the Human Visual System Selectively Sensitive to the Orientation and Size of Retinal Images," *Journal of Physiology, London*, *203*:237–260 (1969).
4. Bush, Larry F. *The Design of an Optimum Alphanumeric Symbol Set for Cockpit Displays*. MS thesis, AFIT/GE/EE/77-11, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1977.
5. Campbell, F. W. and others. "The Effect of Orientation on the Visual Resolution of Gratings," *Journal of Physiology, London*, *187*:427–436 (1966).
6. Churchland, Paul M. and Patricia Smith Churchland. "Could a Machine Think?," *Scientific American*, *262*(1):32–37 (January 1990).
7. Coren, Stanley and Joan Stern Girgus. *Seeing is Deceiving: The Psychology of Visual Illusions*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1978. First chapter provides history of illusions; summarizing chapter adds critique of Ginsburg-type approach.
8. Daugman, John. "Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dimensional Visual Cortical Filters," *Journal of the Optical Society of America*, *2*(7):1160–1169 (July 1985).
9. DeValois, R.L. and others. "Cortical Cells: Bar and Edge Detectors, or Spatial Frequency Filters." In Cool, S.J. and E. L. Smith III, editors, *Frontiers of Visual Science*, New York: Springer-Verlag, 1978.
10. Ehrenstein, Walter. "Modifications of the Brightness Phenomenon of L. Hermann." In Petry, Susan and Glenn E. Meyer, editors, *The Perception of Illusory Contours*, chapter 3, pages 35–39, New York, NY: Springer-Verlag, 1987. This chapter was translated from Uber Abwandlundgen der L. Hermannschen Helliskeitserscheinung, by Ehrenstein, 1941. *Zeitschrigt für Psychologie, 150,* 83-91. Copyright 1941 by Verlag von Johann Ambrosius Barth.
11. Gabor, Dennis. "Theory of Communication," *Journal of IEE*, *93*:429–457 (1946).
12. Ginsburg, Arthur. *Psychological Correlates of a Model of the Human Visual System*. MS thesis, AFIT/GE/EE/71S-2, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1971.

13. Ginsburg, Arthur. *Visual Information Processing Based on Spatial Filters Constrained by Biological Data.* Technical Report AMRL-TR-78-129, Volumes I and II, Aerospace Medical Research Laboratory, December 1978.

14. Ginsburg, Arthur P. "The Relationship Between Spatial Filtering and Subjective Contours." In Petry, Susan and Glenn E. Meyer, editors, *The Perception of Illusory Contours*, chapter 13, pages 126–130, New York, NY: Springer-Verlag, 1987.

15. Grossberg, Stephen and Ennio Mingolla. "The Role of Illusory Contours in Visual Segmentation." In Petry, Susan and Glenn E. Meyer, editors, *The Perception of Illusory Contours*, chapter 12, pages 116–125, Springer-Verlag, 1987.

16. Hubel, David. "Single Unit Activity in Striate Cortex of Unrestrained Cats," *Journal of Physiology, London*, *147*:226–238 (1959).

17. Hubel, David. "The Visual Cortex of the Brain," *Scientific American*, *209*(5):54–62 (November 1963).

18. Hubel, David and T. Wiesel. "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex," *Journal of Physiology, London*, *160*:106–154 (1962).

19. Hubel, David and T. Wiesel. "Receptive Fields and Functional Architecture of Monkey Striate Cortex," *Journal of Physiology, London*, *195*:215–243 (1968). Actual reference taken from page 142, Introduction to Psychology, by Ernest Hilgard, R.L. Atkinson and R.C. Atkinson.

20. Hubel, David and T. Wiesel. "Sequence Regularity and Geometry of Orientation Columns in the Monkey Striate Cortex," *Journal of Comp. Neurology*, *158*:267–293 (1974).

21. Jones, Judson P. and others. "The Two-Dimensional Spectral Structure of Simple Receptive Fields in Cat Striate Cortex," *Journal of Neurophysiology*, *58*(6):1212–1232 (December 1987).

22. Jones, Judson P. and Larry A. Palmer. "An Evaluation of the Two-Dimensional Gabor Filter Model of Simple Receptive Fields in Cat Striate Cortex," *Journal of Neurophysiology*, *58*(6):1233–1258 (December 1987).

23. Jones, Judson P. and Larry A. Palmer. "The Two-Dimensional Spatial Structure of Simple Receptive Fields in Cat Striate Cortex," *Journal of Neurophysiology*, *58*(6):1187–1211 (December 1987).

24. Kabrisky, Matthew. *A Proposed Model for Visual Information Processing in the Human Brain.* Urbana, IL: University of Illinois Press, 1966.

25. Kabrisky, Matthew. "A Proposed Model for Visual Information Processing in the Human Brain." In Wathen-Dunn, W., editor, *Models for the Perception of Speech and Visual Form*, pages 354–361, Cambridge, MA: MIT Press, 1967.

26. Kabrisky, Matthew and others. "A Theory of Pattern Perception Based on Human Physiology," *Ergonomics*, *13*:129–142 (1970).

27. Koehler, W. and H. Wallach. "Figural After-effects: An Investigation of Visual Processes," *Proceedings of the American Philosophical Society*, *88*:269–357 (1944).

28. Kulikowski, J. J. and P. O. Bishop. "Fourier Analysis and Spatial Representation in the Visual Cortex," *Experientia*, *37*:160–163 (1981).

29. MacKay, D. M. "Strife Over Visual Cortical Function," *Nature*, *289*:117–118 (1981).

30. Macleod, I. D. G. and A. Rosenfeld. "The Visibility of Gratings: Spatial Frequency Channels or Bar-Detecting Units?," *Vision Research*, *14*:909–915 (1974).

31. Maffei, L. and A. Fiorentini. "The Visual Cortex as a Spatial Frequency Analyzer," *Vision Research*, *13*:1255–1267 (1973).

32. Marčelja, S. "Mathematical Description of the Responses of Simple Cortical Cells," *Journal of the Optical Society of America*, *70*(11):1297–1300 (November 1980).

33. Merriam and Webster, editors. *Webster's New Collegiate Dictionary*. Springfield, MA: G. & C. Merriam Co., 1981.

34. Messner, Richard A. and Harold H. Szu. "An Image Processing Architecture for Real Time Generation of Scale and Rotation Invariant Patterns," *Computer Vision, Graphics, and Image Processing*, *31*:50–66 (1985).

35. Movshon, J. A. and others. "Spatial Summation in the Receptive Fields of Simple Cells in the Cat's Striate Cortex," *Journal of Physiology, London*, *283*:53–77 (1978).

36. Movshon, J. A. and D. J. Tolhurst. "On the Response Linearity of Neurons in Cat Visual Cortex," *Journal of Physiology, London*, *249*:56P–57P (1975).

37. Petry, Susan and Glenn E. Meyer, editors. *The Perception of Illusory Contours*. New York, NY: Springer-Verlag, 1987. First chapter is a good background of various methods used to study illusory contours.

38. Prazdny, K. "Illusory Contours Are Not Caused By Simultaneous Brightness Contrast," *Perception and Psychophysics*, *34*:403–404 (1983).

39. Rand, Ayn. *For the New Intellectual*. New York, NY: Signet Books, 1961. Text is reprinted from Atlas Shrugged, 1957.

40. Recognition, Pattern, January 1990. Class notes from AFIT course EENG620, Pattern Recognition I.

41. Rogers, Steven K. Informal meeting with Dr. Steven Rogers. Discussion regarded information processing of external data vs. processing of data internal to the brain. Times associated with processing of external data attributed to Dr. Matthew Kabrisky. October 16, 1990.

42. Rogers, Steven K. and Matthew Kabrisky. Weekly thesis meeting with Dr. Steven Rogers and Dr. Matthew Kabrisky. Discussion detailed known and hypothesized activites taking place on the retina, in the lateral geniculate nucleus, and the visual cortex. September 25, 1990.

43. Searle, John R. "Is the Brain's Mind a Computer Program?," *Scientific American*, *262*(1):26–31 (January 1990).

44. Shapely, R. and J. Gordon. "Nonlinearity in the Perception of Form," *Perception and Psychophysics*, *37*:84–88 (1985).

45. Sun Microsystems Inc. *SPARC Release Notes for 4.0*, 1988. Part Number: 800-1742-10. Revision A of 9 May 1988.

46. Sun Microsystems Inc. *TAAC-1 Application Accelerator: Software Reference Manual*, 1988. Part Number: 800-3202-11. Revision A of 15 April 1989. Bible No. 2 regarding TAAC usage–details of all software routines supplied with TAAC.

47. Sun Microsystems Inc. *TAAC-1 Application Accelerator: User Guide*, 1988. Part Number: 800-2177-11. Revision A of 15 September 1988. Bible No. 1 regarding TAAC usage–overview of what makes TAAC go.

48. van de Goor, A.J. *Computer Architecture and Design*. Addison-Wesley Publishing Co., 1989.

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | December 1990 | Master's Thesis |

**4. TITLE AND SUBTITLE**
Analysis of Visual Illusions Using Gabor Filters

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
Richard A. Oberndorf, Captain, USAF

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Air Force Institute of Technology
WPAFB, OH 45433-6583

**8. PERFORMING ORGANIZATION REPORT NUMBER**
AFIT/GE/ENG/90D-47

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** (Maximum 200 words)

This effort has demonstrated the correctness of using spatial filters in the analysis of various visual illusions; the specific form of filter has been derived from a Gabor equation model of simple cell response on the visual cortex. The Gabor Low Pass Filter (GLPF) applied to these anomalies was derived from proposals made that simple cell response on the visual cortex may be modeled by a set of equations originally proposed by Gabor in the 1940's. Based upon the extension of these equations into two dimensions, a GLPF process was applied to computer-generated black & white illusions (the Kanizsa Triangle, the Spoked Circle and the Ehrenstein Illusion). The results demonstrate that the anomalous contour present in these illusions are explained by an energy boundary surrounding the anomalous area. These differing energies are a direct result of the GLPF process.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Illusions; Low Pass Filters; Fourier Transformation; Visual Cortex; Gabor Equations | 147 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |